GROUP EXHIBIT W

US005579517A

United States Patent [19]

Reynolds et al.

[11] **Patent Number: 5,579,517**

[45] **Date of Patent: Nov. 26, 1996**

[54] **COMMON NAME SPACE FOR LONG AND SHORT FILENAMES**

[75] Inventors: **Aaron R. Reynolds**, Redmond; **Dennis R. Adler**, Mercer Island; **Ralph A. Lipe**, Woodinville; **Ray D. Pedrizetti**, Issaquah; **Jeffrey T. Parsons**; **Rasipuram V. Arun**, both of Redmond, all of Wash.

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

[21] Appl. No.: **427,004**

[22] Filed: **Apr. 24, 1995**

Related U.S. Application Data

[63] Continuation of Ser. No. 41,497, Apr. 1, 1993, abandoned.

[51] **Int. Cl.**[6] **G06F 17/30**

[52] **U.S. Cl.** **395/616**; 395/425; 395/500; 395/575; 364/246; 364/DIG. 1; 364/254; 364/952.9; 364/962; 364/DIG. 2

[58] **Field of Search** 395/425, 600, 395/500

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,987,531	1/1991	Nishikado et al.	395/600
5,307,494	4/1994	Yasumatsu et al.	395/600
5,313,646	5/1994	Hendricks et al.	395/600
5,359,725	10/1994	Garcia et al.	395/500
5,371,885	12/1994	Letwin	395/600
5,388,257	2/1995	Bauer	395/600
5,412,808	5/1995	Bauer	395/600
5,421,001	5/1995	Methe	395/500

OTHER PUBLICATIONS

Ray Duncan, "Using long filenames and extended attributes" parts 1 & 2, PC Magazine vol. 9 nos 8 & 9. pp. 317 & 305, Apr. 24 & May 15, 1990.

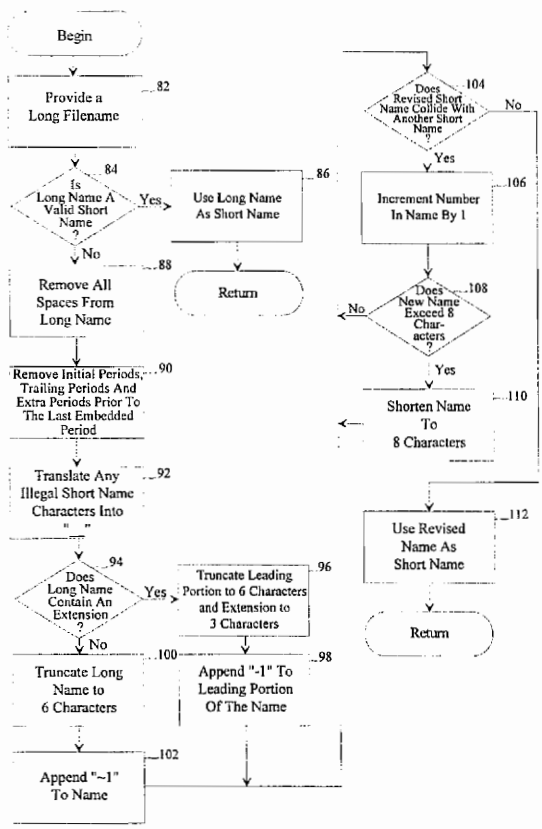
Ray Duncan, "Design Goals and Implementation of the new High Performance File System" Microsoft Systems Journal, vol. 4, No. 5, pp. 1–13, Sep. 1989.

Primary Examiner—Thomas G. Black
Assistant Examiner—C. Pham
Attorney, Agent, or Firm—Seed and Berry

[57] **ABSTRACT**

An operating system provides a common name space for both long filenames and short filenames. In this common namespace, a long filename and a short filename are provided for each file. Each file has a short filename directory entry and may have at least one long filename directory entry associated with it. The number of long filename directory entries that are associated with a file depends on the number of characters in the long filename of the file. The long filename directory entries are configured to minimize compatibility problems with existing installed program bases.

4 Claims, 8 Drawing Sheets



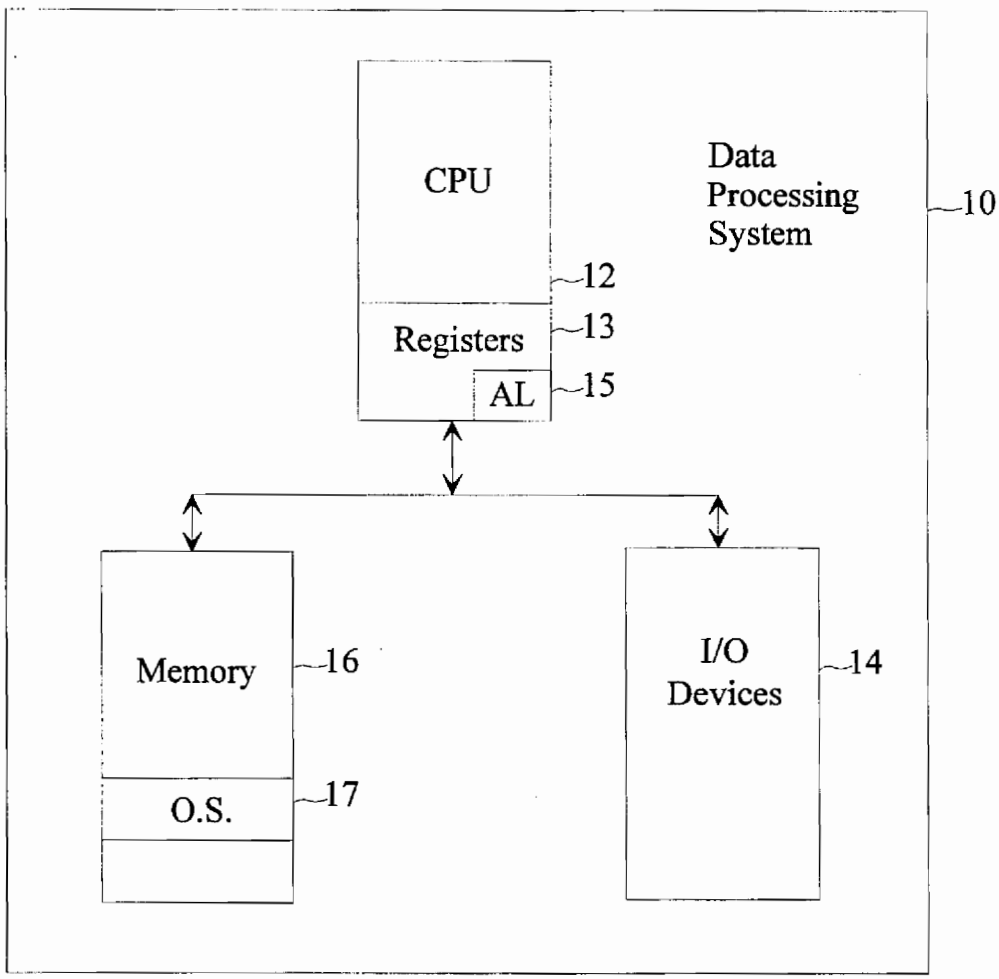

CPU
12
Registers
13
AL
15
Data
Processing
System
10
Memory
16
O.S.
17
I/O
Devices
14

Figure 1

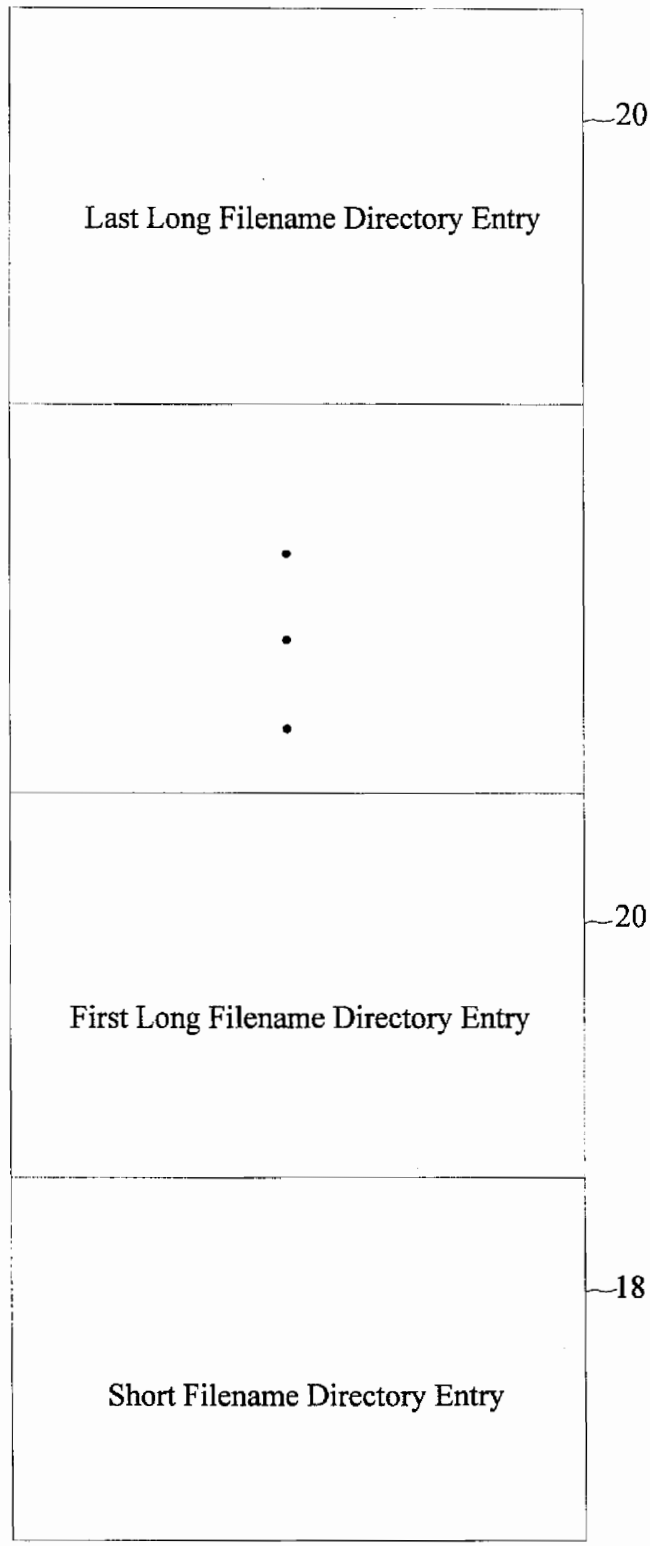

Last Long Filename Directory Entry
20
First Long Filename Directory Entry
20
Short Filename Directory Entry
18

Figure 2

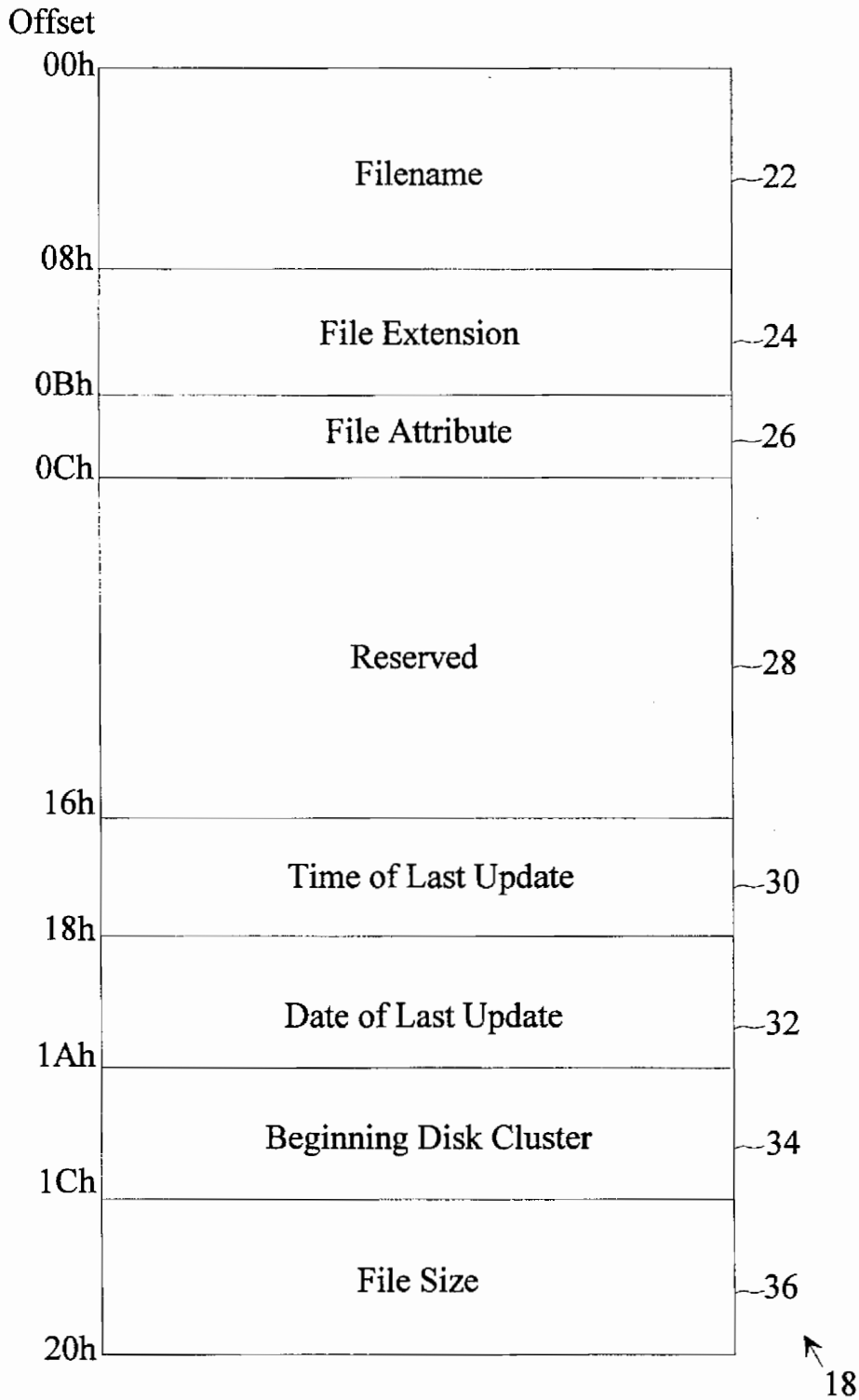

Offset
00h
Filename
22
08h
File Extension
24
0Bh
File Attribute
26
0Ch
Reserved
28
16h
Time of Last Update
30
18h
Date of Last Update
32
1Ah
Beginning Disk Cluster
34
1Ch
File Size
36
20h
18

Figure 3a

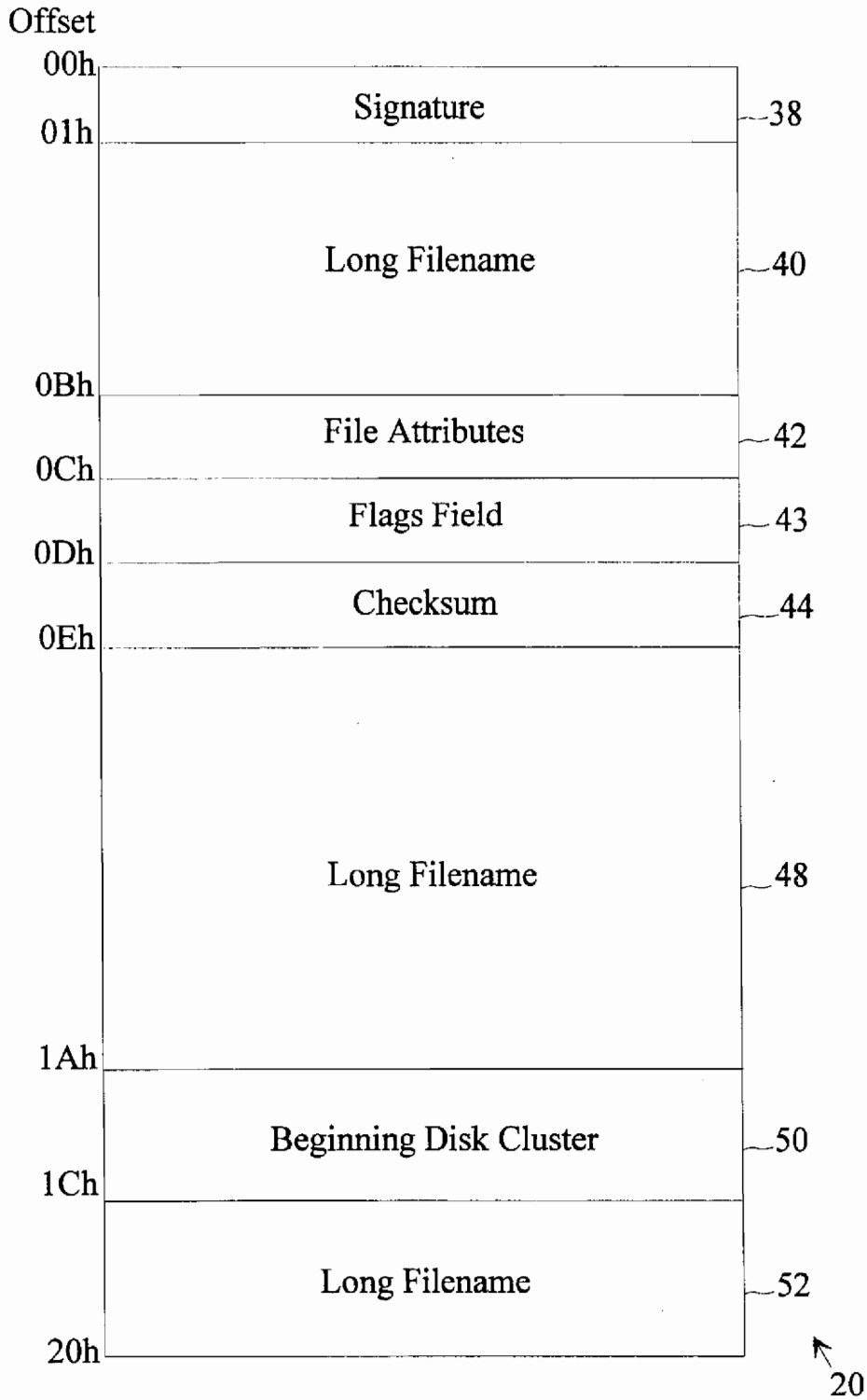

Offset
00h
01h
Signature
38
Long Filename
40
0Bh
File Attributes
42
0Ch
Flags Field
43
0Dh
Checksum
44
0Eh
Long Filename
48
1Ah
Beginning Disk Cluster
50
1Ch
Long Filename
52
20h
20

Figure 3b

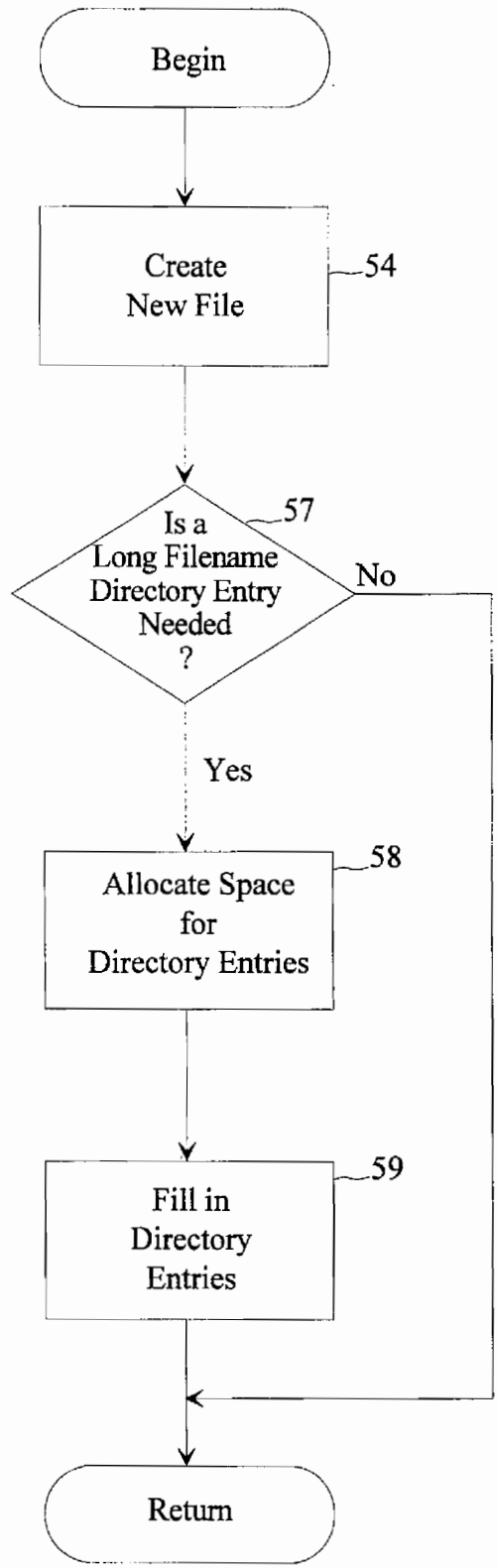

Begin
Create New File
54
Is a Long Filename Directory Entry Needed ?
57
No
Yes
Allocate Space for Directory Entries
58
Fill in Directory Entries
59
Return

Figure 4

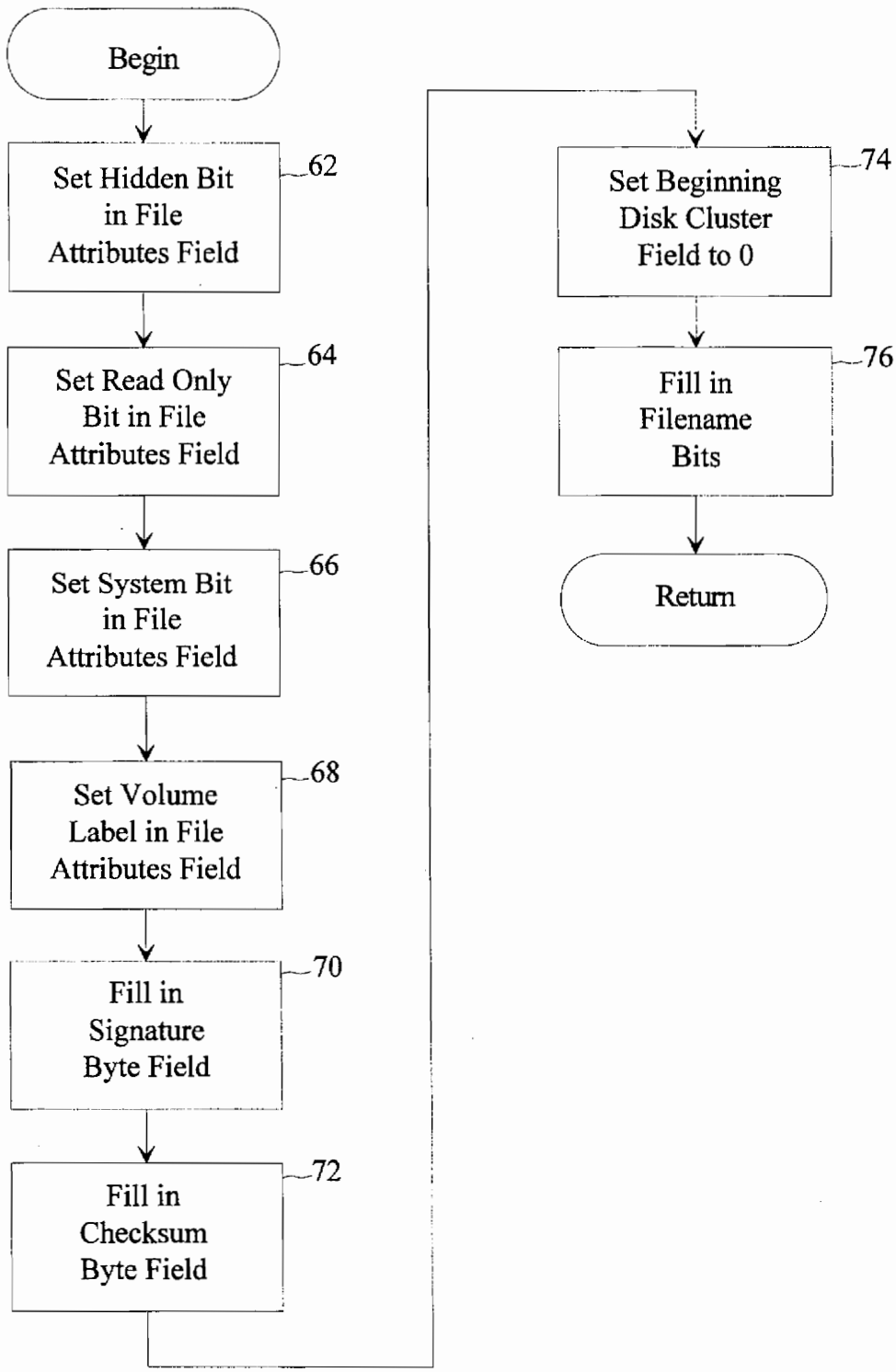

Begin
Set Hidden Bit in File Attributes Field
62
Set Read Only Bit in File Attributes Field
64
Set System Bit in File Attributes Field
66
Set Volume Label in File Attributes Field
68
Fill in Signature Byte Field
70
Fill in Checksum Byte Field
72
Set Beginning Disk Cluster Field to 0
74
Fill in Filename Bits
76
Return

Figure 5a

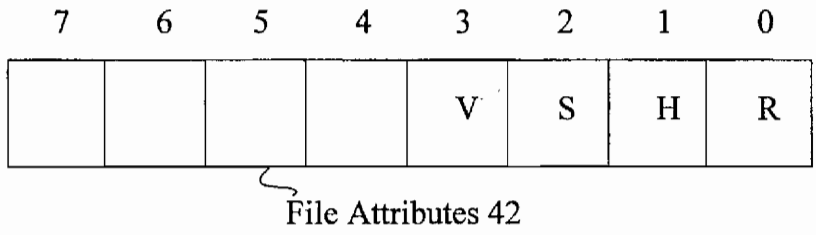


Figure 5b

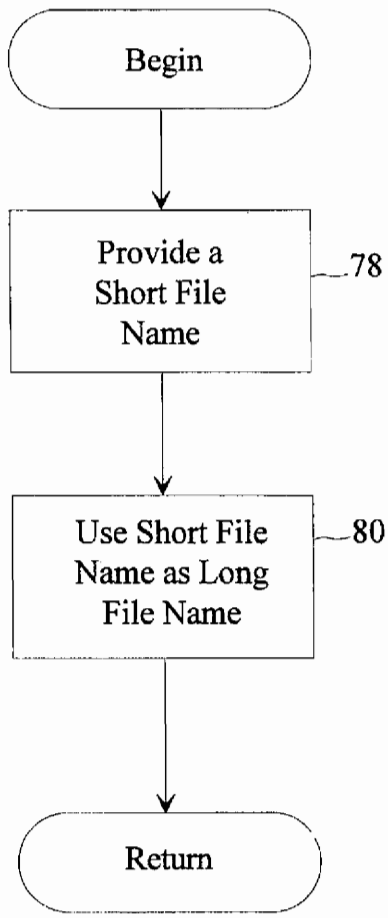


Figure 6a

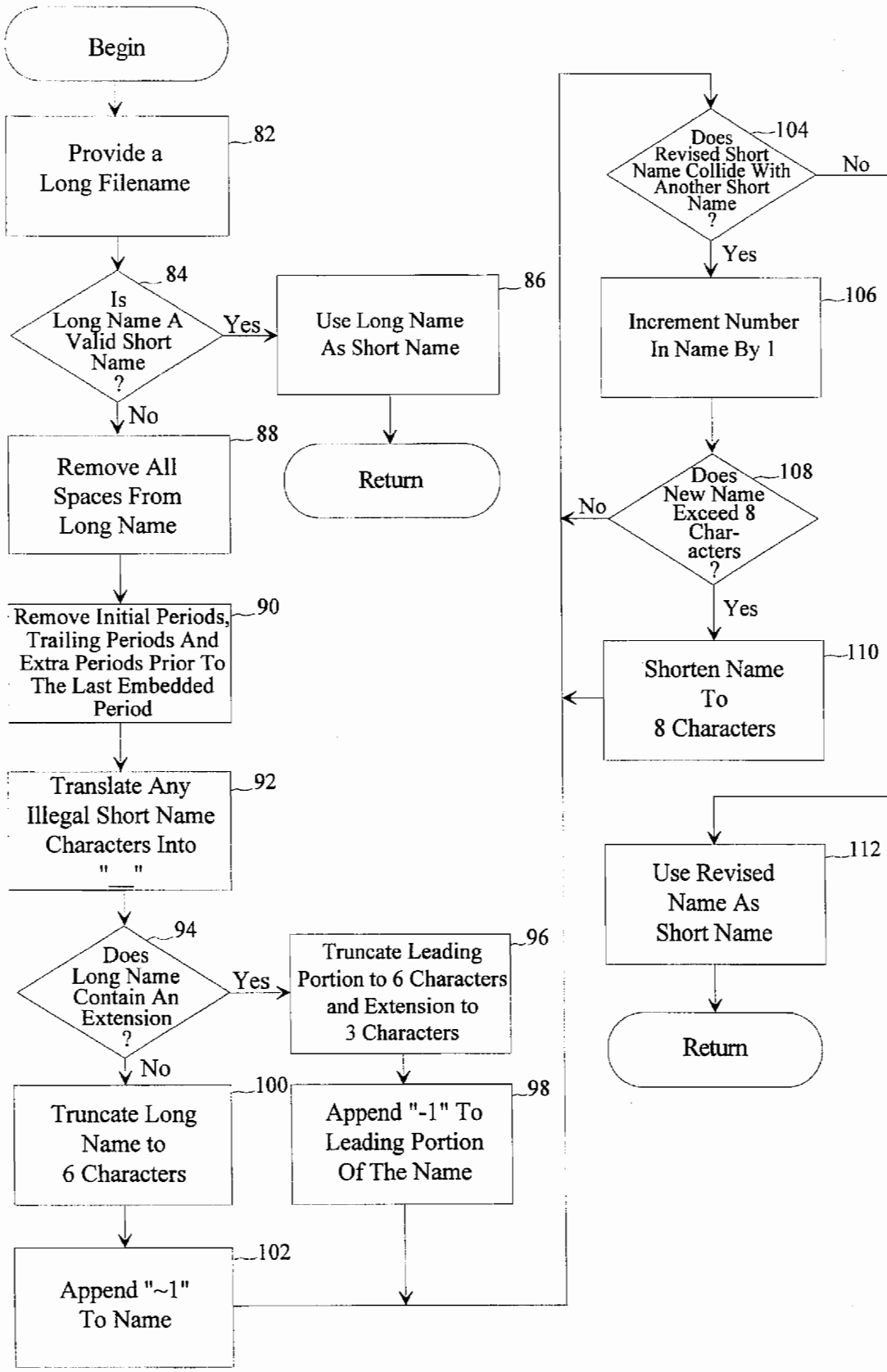

Begin
Provide a Long Filename
82
Is Long Name A Valid Short Name ?
84
Yes
Use Long Name As Short Name
86
Return
No
Remove All Spaces From Long Name
88
Remove Initial Periods, Trailing Periods And Extra Periods Prior To The Last Embedded Period
90
Translate Any Illegal Short Name Characters Into " _ "
92
Does Long Name Contain An Extension ?
94
Yes
Truncate Leading Portion to 6 Characters and Extension to 3 Characters
96
No
Truncate Long Name to 6 Characters
100
Append "-1" To Leading Portion Of The Name
98
Append "~1" To Name
102
Does Revised Short Name Collide With Another Short Name ?
104
No
Yes
Increment Number In Name By 1
106
Does New Name Exceed 8 Char-acters ?
108
No
Yes
Shorten Name To 8 Characters
110
Use Revised Name As Short Name
112
Return

Figure 6b

COMMON NAME SPACE FOR LONG AND SHORT FILENAMES

CROSS-REFERENCE TO RELATED APPLICATION

This application is a continuation of U.S. patent application Ser. No. 08/041,497, filed Apr. 1, 1993, now abandoned.

TECHNICAL FIELD

The present invention relates generally to data processing systems and, more particularly, to a common name space for long and short filenames.

BACKGROUND OF THE INVENTION

Many operating systems, such as the MS-DOS, version 5, operating system, sold by Microsoft Corporation of Redmond, Washington, support only short filenames. In the MS-DOS, version 5, operating system, filenames may be a maximum length of eleven characters. Each filename may have a main portion of eight characters followed by an extension of three characters. An example filename in the MS-DOS, version 5, operating system is "EXAMPLE1.EXE", wherein "EXAMPLE1" constitutes the main portion and "EXE" constitutes the extension.

Each filename is limited to eleven characters due to constraints in the file system of the MS-DOS, version 5, operating system. This file system employs a directory structure in which each file has a directory entry associated with it. Unfortunately, the directory entry for a file only supports filenames with a maximum length of eleven characters. Such a limit in the length of the filenames is often frustrating to a user. The length limit of eleven characters prevents a user from employing adequately descriptive filenames and, in many instances, forces a user to insert awkward abbreviations of descriptive terms into the filename.

SUMMARY OF THE INVENTION

It is, therefore, an object of the present invention to provide a system that supports long filenames.

It is another object of the present invention to provide a system that supports long filenames while minimizing the compatibility impact of supporting long filenames.

It is a further object of the present invention to provide a system that supports a common name space for both long filenames and short filenames.

In accordance with the first aspect of the present invention, a method is practiced in a data processing system having a memory means and a processing means. In accordance with this method, a first directory entry is created and stored in the memory means for a file. The first directory entry holds a first filename for the file and information about the file. A second directory entry is also created and stored in the memory means. The second directory entry holds at least one portion of a second filename having a fixed number of characters and information about the file. One of the first or second directory entries is accessed in the memory means to gain access to the information contained therein.

In accordance with another aspect of the present invention, a data processing system includes a memory that holds a first directory entry for a file, a second directory entry for the file, and an operating system. The first directory entry includes a first filename for the file and the second directory entry includes the second filename for the file. The second filename includes more characters than the short filename. The data processing system also includes a processor for running the operating system and accessing either the first directory entry or the second directory entry to locate the file.

In accordance with yet another aspect of the present invention, a method is practiced in a data processing system having memory. In accordance with this method, a file is created and the file is assigned a user-specified long filename. The long filename is manipulated with the data processing system to create a short filename of fewer characters. The long filename and the short filename are stored in memory so that the file can be accessed by either the long filename or the short filename.

In accordance with a further aspect of the present invention, a method is practiced in which a first directory entry for a file is stored in a memory means. The first directory entry holds the short filename for the file. The short filename includes at most a maximum number of characters that is permissible by an application program. A second directory entry is also stored in the memory means for the file. A second directory entry holds at least the first portion of a long filename for the file. The long filename includes a greater number of characters than the maximum number of characters that is permissible by the application program. The application program is run on a processor of the data processing system. The application program identifies the file by the short filename.

In accordance with a still further aspect of the present invention, a method is practiced in which a first directory entry is stored in the memory means for a file. The first directory entry holds a short filename for the file that includes at most the maximum number of characters that is permissible by the operating system. A second directory entry is stored in the memory means for the file. The second directory entry holds a long filename for the file that includes more than the maximum number of characters that is permissible by the operating system. In this instance, the operating system does not use long filenames; rather, it uses solely short filenames. The first directory entry is accessed by the operating system to locate the file.

BRIEF DESCRIPTION OF THE DRAWINGS

A preferred embodiment of the present invention will now be described herein with reference to the Drawings. The Drawings include the following Figures.

FIG. **1** is a block diagram of a data processing system used for implementing the preferred embodiment of the present invention.

FIG. **2** is a block diagram illustrating the storage of short filename directories in locations adjacent to long filename directory entries.

FIG. **3***a* shows the format of a short filename directory entry in the preferred embodiment of the present invention.

FIG. **3***b* shows the format of a long filename directory entry in the preferred embodiment of the present invention.

FIG. **4** is a flow chart illustrating the steps performed by the preferred embodiment of the present invention when a new file is created.

FIG. **5***a* is a flow chart illustrating the steps performed in creating a long filename directory entry in the preferred embodiment of the present invention.

FIG. **5***b* is a block diagram illustrating bits in the file attributes fields of the long filename directory entry of FIG. **3***b*.

FIG. *6a* is a flow chart illustrating the steps performed when a short filename is provided by the user in the preferred embodiment of the present invention.

FIG. *6b* is a flow chart illustrating the steps performed when a long filename is provided by user in the preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

A preferred embodiment of the present invention described herein provides support for the use of long filenames (i.e., filenames that may have substantially more characters than current operating systems, such as the MS-DOS, version 5, operating system permit). "Short filenames" will be used hereinafter to refer to filenames that have a small limit (such as 11 characters) as to the maximum number of characters permitted. In the preferred embodiment, the long filenames are provided in a common name space with the short filenames. A long filename and a short filename are provided for each file in the system. The sharing of a common name space is realized through providing separate directory entries for long filenames and short filenames. Each file has a short filename directory entry associated with it and may also have at least one long filename directory entry. The short filenames are like those provided previously in the MS-DOS, version 5, operating system. The long filenames, as will be described in more detail below, may have a maximum length of up to 255 characters. The preferred embodiment will be described with reference to an implementation with the MS-DOS, version 5, operating system.

The potential compatibility problems of supporting long filenames are apparent by considering one solution to the problem of short filenames. This solution is not part of the present invention and is described herein merely to illustrate how a preferred embodiment avoids the compatibility problems suffered by this proposed solution. This solution supports long filenames by merely increasing the number of characters the operating system permits for a filename.

There are two major difficulties with this solution. First, the existing application bases of many systems use only short filenames (e.g., 11 characters or less) and are not prepared to utilize only long filenames (e.g., up to 255 characters). As an example, an application may allocate a buffer large enough to hold the short filename and if the operating system tries to place long filename data into this buffer, the buffer may overflow so as to cause the application data to be unexpectedly overwritten. Second, certain disk utility programs access the file system volume directly and, thus, do not rely on the operating system to access the files. If the file system is changed to support long filenames, compatibility problems with the disk utility programs arise.

The preferred embodiment of the present invention described herein, in contrast, seeks to minimize the compatibility impact of supporting long filenames by providing both a long filename and a short filename for each file. As a result, applications and utilities that require short filenames still have short filenames available, and applications that use long filenames have long filenames available.

The preferred embodiment of the present invention may be implemented as code realized as software or firmware. In order to support long filenames, the preferred embodiment of the present invention provides several long filename application program interfaces (APIs). These APIs are provided along with the conventional short filename interfaces that are standard with the operating system. The long filename APIs support file operations and directory entries for long filenames. The APIs include a file attributes function, a file delete function, a file directory function, a file find function, a file open/create function and a file rename function.

The preferred embodiment of the present invention may be implemented in a data processing system **10** like that shown in FIG. **1**. This data processing system **10** includes a central processing unit (CPU) **12** with a standard set of registers **13** that includes an accumulator (AL) register **15**, a memory **16** and input/output (I/O) devices **14**. The CPU **12** oversees the operations of the data processing system **10** and has access to the memory **16** and the I/O devices **14**. The memory **16** may include both RAM and disc storage. The memory **16** holds an operating system **17** (denoted as "O.S." in FIG. **1**) which includes the long and short filename APIs. Those skilled in the art will appreciate that the present invention may be implemented on other suitable data processing systems.

All of the functions for the long filename APIs and short filename APIs are incorporated into the operating system **17**. Those functions are supported through an Int **21***h* interrupt call (where **21***h* denotes **21** in hexadecimal notation). In other words, all the functions are called by executing an Int **21***h* interrupt, wherein the function that is called through the Int **21***h* interrupt is specified by a value placed in a register, as will be described in more detail below. The Int **21***h* interface is like that provided in the MS-DOS, version 5, operating system except that the interface also supports calls to functions for long filenames. In calls to the long filename APIs, the function number to be called is placed in the AL register **15** of a processor, such as the CPU **12** in FIG. **1** before the interrupt is initiated.

In order to support both a long filename and a short filename for each file, the preferred embodiment provides a short filename directory entry **18** (FIG. **2**) and may provide at least one long filename directory entry **20** for each file in a common name space. Each file has a long filename and a short filename associated with it. A long filename directory entry **20** is only created when the long filename cannot be correctly stored in the short filename directory entry. The long filename directory entries **20** are stored adjacent to the corresponding short filename directory entry **18** as part of the common name space used in memory **16**. Moreover, the long filename directory entries **20** are configured to minimize compatibility problems with operating systems that support only short filenames.

FIG. **2** shows an example of the directory entries **18** and **20** for a file in the preferred embodiment described herein. The short filename directory entry **18** is provided along with several long filename directory entries **20**. The number of long filename directory entries **20** provided (including zero long filename directory entries) for a file depends upon the number and type of characters in the long filename. As will be described in more detail below, each long filename directory entry **20** may hold up to 26 characters of a long filename. The long filename directory entries **20** are dynamically allocated based upon the number of characters in the long filename. For example, a file with a long filename of 50 characters has two long filename directory entries **20** allocated for it, whereas a file with a long filename of 70 characters has three long filename directory entries **20** allocated for it. As was mentioned above, a long filename may have a maximum of 255 characters and thus, a maximum of 10 long filename directory entries **20** may be allocated for any file. The maximum of 255 characters per

filename is a product of maximum path length (260 characters) limitations of the operating system **17**.

There may be many instances in which the long filename does not completely fill all of the space available in the allocated long filename directory entries **20**. In such an instance, a null terminator is placed after the last character of the long filename so that additional spaces or nonsensical data will not be returned. The extra spaces are filled with **0**FFh (where "h" indicates the use of hexadecimal notation).

FIG. **3***a* illustrates the format of the short filename directory entry **18**. Each of the fields in the directory entry begins at a different offset relative to the starting address of the directory entry. A filename field **22** holds the main portion (i.e., the leading 8 characters) of the short filename. As the main portion of the short filename may hold up to eight characters of the short filename, the filename field **22** is eight bytes in length and begins at offset 00h. The filename field **22** is followed by a file extension field **24** at offset 08h. The file extension field holds the characters of the extension of the short filename. The extension field **24** is three bytes in length (encoding three characters).

Following the extension field **24** at offset 08h is a file attributes field **26**. The file attributes field **26** includes a number of bits that, based upon whether the bits are set or not, specify information about the associated file.

The short filename directory entry **18** also includes a reserved field **28**. The reserved field **28** begins at offset 0Ch and is ten bytes in length. The short filename directory entry **18** additionally includes a time of last update field **30** and a date of last update field **32**. The time of last update field **30** is two bytes in length and begins at offset 16h. The date of last update field **32** is two bytes in length and begins at offset 18h.

The short filename directory entry **18** includes a beginning disk cluster field **34**. The beginning disk cluster field **34** holds a pointer to the section of the memory **16** (FIG. 1) where the file's first disk cluster is held (i.e. to the beginning of the allocation chain for the file). This beginning disk cluster field **34** (FIG. **3***a*) is stored at offset 1Ah and is two bytes in length. A file size field **36** follows the beginning disk cluster field **34**. The file size field **36** holds a value that specifies the amount of memory occupied by the file associated with the short filename directory entry **18**. The file size field **36** is four bytes in length and begins at offset 1Ch.

FIG. **3***b* illustrates the format used for each of the long filename directory entries **20**. The long filename directory entry **20** additionally includes a signature field **38** that holds a digital signature. The signature field **38** is useful in specifying the order of a long filename directory entry **20** in a sequence of associated long filename directory entries. For example, a first long filename directory entry includes a signature field **38** that specifies that it is the first entry, and each successive long filename directory entry includes a signature field **38** that specifies where the long filename directory entry fits in the sequence of long filename directory entries for a file. The signature field **38** is provided primarily for use with utility programs that directly access the file system volume. The signature field **38** is one byte in length and begins at offset 00h, which is the beginning of the filename field **22** (FIG. **3***a*) of the short filename directory entry **18**. The signature field **38**, given its location in the long filename directory entry, might easily be mistaken for a portion of a short filename by certain utility programs. Hence, the signature field **38** includes only illegal short filename characters so that the characters may not be readily changed by systems or utilities that support only short filenames.

The long filename directory entry **20** includes three fields **40**, **48** and **52** that are provided for holding characters of the long filename. The first long filename field **40** begins at offset 01h and may hold up to ten characters of the long filename (i.e., it is 10 bytes in length). The second long filename field **48** begins at offset **0**Eh and may hold up to twelve characters (i.e., 12 bytes) of the long filename. Lastly, the third long filename field **52** begins at offset 1Ch and may hold up to four characters (i.e., 4 bytes) of the long filename. Thus, cumulatively, these three fields **40**, **48** and **52** may hold up to twenty-six characters of the long filename. The long filename fields **40**, **48** and **52** are filled sequentially beginning with field **40** and then filling fields **48** and **52**, consecutively.

While the long filename directory entry **20** differs from the short filename directory entry **18**, the long filename directory entry **20**, nevertheless, includes certain similar fields at the same specified offsets as were discussed above for the short filename directory entry **18** (FIG. **3***a*). As such, operating systems that do not support long filenames are not disturbed by the long filename directory entries **20**. For instance, the long filename directory entry **20** includes a file attributes field **42** which is like the file attributes field **26** (see FIG. **3***a*) provided in the short filename directory entry.

The long filename directory entry **20** contains a checksum field **44**, which is one byte in length and at offset 0Dh. The checksum field **44** holds a checksum of the short filename. The checksum byte, as will be described in more detail below, is used to ensure that the long name is valid for the associated short filename and to act as a pointer to the short filename directory entry **18** that is helpful to disk utility programs. A flags field **43** is held at offset 0Ch. The flags field **43** holds a flag bit that may be set when unicode characters are used.

In addition, the beginning disk cluster field **50** (FIG. **3***b*) of the long filename directory entry **20** is analogous to the beginning disk cluster field **34** (FIG. **3***a*) of the short filename directory entry **18**. However, it always has a constant value of zero in the long filename directory entry.

The above discussion has focused on how the directory entries **18** and **20** (FIG. **2**) are used to support both long filenames and short filenames. The discussion below will focus on how such directory entries are supported by the preferred embodiment of the present invention.

When a new file is created, the preferred embodiment must take steps to support both a long filename and a short filename for the new file. In discussing how the preferred embodiment supports both long filenames and short filenames, it is helpful to first focus on the creation of the directory entries and then to focus on the creation of the filenames. FIG. **4** is a flowchart depicting the basic steps performed upon creation of the new file. Initially, the new file is created (step **54**) using either a long filename API or a short filename API. Both varieties of APIs support the creation of files. Depending on the type of API that is used to create the files, the file will initially have a long filename and/or a short filename. In other words, if a file is created with a long filename API, it will initially have a long filename and if a file is created with a short filename API, it will initially have a short filename, which may also be the long filename for the file.

At least one long filename directory entry **20** may be created for the file. First, a determination is made whether a long filename directory entry **20** is required (step **51**). If the long filename will not correctly fit in the short filename directory entry **18**, a long filename directory entry **20** is

required. Long filename directory entries **20** are dynamically allocated based upon the number of characters in the long filename. At a minimum, a short filename directory entry **18** will be created that has the format that is shown in FIG. **3***a*. Thus, the system checks to see how many long filename directory entries are needed and allocates space for the short filename directory entry and as many additional long filename directory entries as are required (step **58**). It should be appreciated that when both a short filename directory entry **18** and at least one long filename directory entry **20** are created, space for both types of directory entries are allocated together at the same time. The long and short filename directory entries **18** and **20** are then filled in step **59**. However, if no long filename directory entry is required, no space will be allocated (i.e., steps **58** and **59** are skipped).

FIG. **5***a* is a flowchart depicting the steps performed in filling in a long filename directory entry **20** (see step **59** in FIG. **4**). The steps are shown in a given sequence, but those skilled in the art will appreciate that the steps need not be performed in this illustrated sequence. Rather, other sequences are equally acceptable.

A hidden bit in the file attributes field **42** is set to have a value of one (step **62**). FIG. **5***b* shows the bits included in the file attributes field **42**. The hidden bit is designated by the letter "H" in FIG. **5***b* and is present at bit position **1** in the file attributes field **42**. When the hidden bit is set to a value of one, the directory entry is hidden and is excluded from normal searches of the directory entries **18** and **20**. By setting the hidden bit, the long filename directory entries **20** (FIG. **2**) are not searched in conventional directory entry searches. The hidden bit is set so that down level systems (i.e., systems that support only short filenames) will not see the long filename directory entries **20**.

A read-only bit is also set in the file attributes field (step **64** in FIG. **5***a*). The read-only bit is designated by the letter "R" in FIG. **5***b* and is present at bit position **0** in the file attributes field **42**. Setting the read-only bit to a value of one indicates that the file is a read-only file and any attempts to write to the file will fail.

A system bit in the file attributes field **42** is set to a value of one (step **66** in FIG. **5***a*). The system bit is designated by the letter "S" in FIG. **5***b* and is present at bit position **2** in the file attributes field **42**. Setting the system bit to a value of one designates the file as a system file and excludes the directory entry from normal searches of the directory entries **18** and **20**. The setting of the system bit to a value of one hides the long filename directory entries **20** from down level operating systems that support only short filenames.

Next, a volume label bit is set in the file attributes field **42** (step **68** in FIG. **5***a*). The volume label bit is designated by the letter "V" in FIG. **5***b* and is present at bit position **3** in the file attributes field **42**. Setting the volume label bit to a value of one hides the long filename directory entry from "Check Disk" operations of certain disk utility programs. For example, MS-DOS, version 5.0, includes a utility named CHKDSK. The setting of the volume label attribute hides the long filename directory entries from CHKDSK.

The discussion will now return again to the flowchart of FIG. **5***a*. The signature byte field **38** (FIG. **3***b*) is filled with a digital signature (step **70** in FIG. **5***a*). As was mentioned above, the signature distinguishes the order of the long filename directory entries **20** for the file. The checksum field **44** in FIG. **3***b* is filled with the appropriate checksum of the short filename (step **72** in FIG. **5***a*). The checksum byte field **44** (FIG. **3***b*) is used to associate the long filename directory entries **20** with their appropriate short filename by holding a checksum of the short filename. The beginning disk cluster field **50** (FIG. **3***b*) is set to zero (step **74** in FIG. **5***a*). The long filename directory entry **20**, thus, has no data allocated to it. This helps to make the long filename directory entry invisible in down level systems. Lastly, the bits for the characters of the long filename are stored in the appropriate long filename fields **40**, **48** and **52** (FIG. **3***b*) of the long filename directory entry **20** (step **76** in FIG. **5***a*).

By setting the file attributes field **42** (FIG. **5***b*) bits as described above and by setting the beginning disk cluster field **50** to zero (FIG. **3***b*), the preferred embodiment of the present invention makes the long filename directory entries nearly invisible to operating systems that support only short filenames (i.e., down level systems). Nevertheless, files with long filenames are still permitted in down level operating systems. The long filename directory entries are not visible in the directory entry listing for down level systems. The combination of bit settings in the file attributes field and the zeroing of the beginning disk cluster field **50** make the long filename directory entries invisible to down level systems. Thus, compatibility problems arising from having long filenames in the down level operating system are minimized. Moreover, utility programs, that may skew the order of directory entries, are not a problem. The signature field **40** (FIG. **3***b*) and the checksum field **44** may be used in conjunction to rearrange entries that are out of order. In particular, the checksum fields **44** are used to associate long filename directory entries **20** with a short filename directory entry and the signature fields **40** of the long filename directory entries are used to assign related long filename directory entries into proper sequence.

The discussion above has noted that filenames are created using either short filename APIs or long filename APIs. As a result, when a file is created it has either a long filename or short filename assigned to it by the user, depending on whether a long filename API or short filename API is used. The preferred embodiment of the present invention described herein automatically creates the missing short filename or long filename. For instance, if a file is created using a short filename API, the preferred embodiment described herein establishes a corresponding long filename (which is the same as the short filename). Analogously, if a file is created using a long filename API, the preferred embodiment generates a corresponding short filename that is stored in a short filename directory entry **18**. FIG. **6***a* shows the steps performed by the preferred embodiment when the short filename is provided by the user. In particular, the user provides a short filename (step **78** in FIG. **6***a*), and the short filename is used as the long filename (step **80**). When the user provides a short filename, the system checks whether the name is a valid short filename and whether there are any existing files that pose a conflict (not shown). If there is no problem in terms of format or conflict, the file is assigned the provided short filename. The short filename is then used as the long filename, and there is no long filename directory entry **20** for the file.

When a file is created using a long filename API, the resulting creation of a corresponding short filename may be quite complex. FIG. **6***b* is a flowchart illustrating the steps performed to create the short filename in such an instance. Initially, the long filename is provided by the user (step **82** in FIG. **6***b*). The preferred embodiment then checks whether the long filename is a valid short filename (step **84**). If the long filename is a valid short filename, the long filename is used as the short filename (step **86**).

However, if the long filename does not qualify as a valid short filename, a short filename is created by removing the

spaces from the long filename and using the resulting characters as a proposed short filename (step **88**). Initial periods, trailing periods and extra periods that are prior to the last embedded period are then removed from the proposed short filename (step **90**). Furthermore, any illegal short filename character is translated into an underscore (step **92**). A check of whether the proposed short filename contains an extension is then performed (step **94**). If the proposed short filename contains an extension, the leading main portion of the filename is truncated to six characters in length, and the leading three characters of the extension are used (step **96**). Subsequently, a "~1" is appended to the leading portion of the remaining characters (step **98**) to serve as the short filename.

If the modified long filename does not contain an extension (step **94**), the long filename is truncated to six characters (step **100**), and "~1" is appended to the truncated filename (step **102**) to serve as the short filename. In both of the above-described instances (i.e., the "yes" instance and "no" instance of step **94**), the preferred embodiment next checks whether the proposed short filename collides with any other short filename (step **104**). If the proposed short filename does not collide with another short filename (i.e., there is no other identical short filename), the proposed short filename is assigned as the short filename for the file (step **112**). In the case where the proposed short filename collides with another short filename, the characters that are appended to the name are incremented by one (step **106**) Thus if the number value is initially "~1" the number value is incremented in step **106** by one to "~2". The preferred embodiment checks whether the new proposed short filename exceeds eight characters in length (step **108**). If the new proposed short filename does not exceed eight characters in length, the checking of whether the proposed short filename collides with another short filename is repeated (step **104**). When the number of characters in the filename exceeds eight characters in length, the new short filename is shortened to eight characters (step **110**). In particular, if the length of the leading portion of the filename (ignoring the extension) plus the tilda and the number exceeds eight characters, the leading portion of the filename is shortened until the new proposed short filename (absent the extension) fits in eight characters. For example, the filename "MonKey~10.EXE" is shortened to "MonKe~10.EXE." The above-described steps **104**, **106**, **108** and **110** are repeated until a short filename is created for the file that is of proper length and that does not collide with another short filename.

The preferred embodiment of the present invention provides a solution to the problem of short filenames while minimizing the compatibility impact of the solution. The use of a common name space that provides a long filename and a short filename for each file allows the files to be used both with applications that support short filenames and applications that support long filenames.

While the present invention has been described with reference to a preferred embodiment thereof, those skilled in the art will appreciate that various changes in scope and form may be made without departing from the present invention as defined in the appended claims.

We claim:

1. In a computer system having a processor running an operating system and a memory means storing the operating system, a method comprising the computer-implemented steps of:

(a) storing in the memory means a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters that is permissible by the operating system;

(b) storing in the memory means a second directory entry for a the file wherein the second directory entry holds a long filename for the file and wherein the second directory entry includes an attributes field which may be set to make the second directory entry invisible to the operating system and the step of storing the second directory entry further comprises the step of setting the attributes field so that the second directory entry is invisible to the operating system, said long filename including more than the maximum number of characters that is permissible by the operating system; and

(c) accessing the first directory entry with the operating system.

2. In a computer system having a processor running an operating system and a memory means storing the operating system, a method, comprising the computer-implemented steps of:

(a) storing in the memory means a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters that is permissible by the operating system;

(b) storing in the memory means a second directory entry for the file wherein the second directory entry holds a long filename for the file and storing a checksum of the short filename in the second directory entry, said long filename including more than the maximum number of characters that is permissible by the operating system; and

(c) accessing the first directory entry with the operating system.

3. In a computer system having a processor running an operating system and a memory means storing the operating system, a method, comprising the computer-implemented steps of:

(a) storing in the memory means a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters that is permissible by the operating system;

(b) storing in the memory means a second directory entry for the file wherein the second directory entry holds a long filename for the file, said long filename including more than the maximum number of characters that is permissible by the operating system;

(c) accessing the first directory entry with the operating system;

(d) storing in the memory means at least one additional directory entry holding a next portion of the long filename and a checksum of the short filename.

4. In a computer system having a processor running an operating system and a memory means storing the operating system, a method, comprising the computer-implemented steps of:

(a) storing in the memory means a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters that is permissible by the operating system;

(b) storing in the memory means a second directory entry for the file wherein the second directory entry holds a long filename for the file, said long filename including more than the maximum number of characters that is permissible by the operating system;

(c) accessing the first directory entry with the operating system;

(d) storing in the memory means at least one additional directory entry holding a next portion of the long filename and a signature that uniquely identifies which portion of the long filename.

* * * * *

US005579517C1

(12) EX PARTE REEXAMINATION CERTIFICATE (5616th)

United States Patent

Reynolds et al.

(10) Number: US 5,579,517 C1

(45) Certificate Issued: *Nov. 28, 2006

(54) **COMMON NAME SPACE FOR LONG AND SHORT FILENAMES**

(75) Inventors: **Aaron R. Reynolds**, Redmond, WA (US); **Dennis R. Adler**, Mercer Island, WA (US); **Ralph A. Lipe**, Woodinville, WA (US); **Ray D. Pedrizetti**, Issaquah, WA (US); **Jeffrey T. Parsons**, Redmond, WA (US); **Rasipuram V. Arun**, Redmond, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

Reexamination Request:
No. 90/007,007, Apr. 19, 2004
No. 90/007,371, Jan. 8, 2005

Reexamination Certificate for:
Patent No.: **5,579,517**
Issued: **Nov. 26, 1996**
Appl. No.: **08/427,004**
Filed: **Apr. 24, 1995**

(*) Notice: This patent is subject to a terminal disclaimer.

Related U.S. Application Data

(63) Continuation of application No. 08/041,497, filed on Apr. 1, 1993, now abandoned.

(51) **Int. Cl.**
G06F 12/00 (2006.01)
G06F 17/30 (2006.01)

(52) **U.S. Cl.** **707/200**
(58) **Field of Classification Search** 707/200
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,058,672	A *	11/1977	Crager et al.	370/394
4,780,821	A	10/1988	Crossley	718/100
4,945,475	A	7/1990	Bruffey et al.	707/1
4,945,476	A	7/1990	Bodick et al.	600/301
4,987,531	A	1/1991	Nishikado et al.	707/200
4,999,766	A	3/1991	Peters et al.	707/10
5,058,000	A	10/1991	Cox et al.	707/10
5,083,264	A *	1/1992	Platteter et al.	714/5
5,129,088	A	7/1992	Auslander et al.	711/1
5,179,703	A	1/1993	Evans	717/122
5,202,982	A	4/1993	Gramlich et al.	707/2
5,202,983	A	4/1993	Orita et al.	707/4
5,287,502	A	2/1994	Kaneko	707/4
5,291,595	A	3/1994	Martins	707/200
5,307,494	A *	4/1994	Yasumatsu et al.	707/200
5,313,646	A	5/1994	Hendricks et al.	707/101

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0 462 587 B1	12/1996
EP	0 578 205 B1	3/2000
EP	0 618 540 B1	12/2001
JP	64-41039	2/1989
JP	1 315 843	12/1989
JP	2 148 341	7/1990
JP	4-297 934	10/1992
JP	6 019 763	1/1994

OTHER PUBLICATIONS

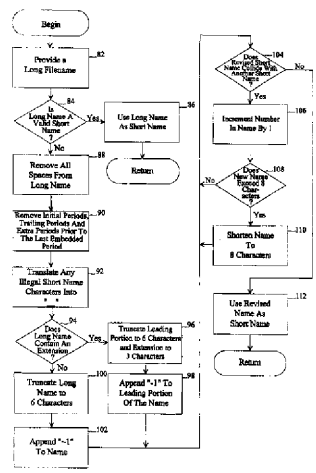
Duncan, Ray Using Long Filenames and Extended Attributes, Part 2, (Power Programming Column Tutorial), PC Magazine, v9, n9, May 15, 1990 ('Duncan') p. 305 (5).*

(Continued)

Primary Examiner—Luke S Wassum

(57) **ABSTRACT**

An operating system provides a common name space for both long filenames and short filenames. In this common namespace, a long filename and a short filename are provided for each file. Each file has a short filename directory entry and may have at least one long filename directory entry associated with it. The number of long filename directory entries that are associated with a file depends on the number of characters in the long filename of the file. The long filename directory entries are configured to minimize compatibility problems with existing installed program bases.



U.S. PATENT DOCUMENTS

5,317,733	A		5/1994	Murdock	707/203
5,329,427	A		7/1994	Hogdahl	361/730
5,355,497	A	*	10/1994	Cohen-Levy	707/200
5,359,724	A		10/1994	Earle	707/205
5,359,725	A	*	10/1994	Garcia et al.	707/200
5,363,487	A	*	11/1994	Willman et al.	710/8
5,367,671	A	*	11/1994	Feigenbaum et al.	707/1
5,371,885	A		12/1994	Letwin	707/205
5,388,257	A		2/1995	Bauer	707/1
5,412,808	A		5/1995	Bauer	707/1
5,421,001	A		5/1995	Methe	707/1
5,434,974	A		7/1995	Loucks et al.	707/101
5,437,029	A		7/1995	Sinha	707/200
5,483,652	A		1/1996	Sudama et al.	707/10
5,485,606	A		1/1996	Midgdey et al.	707/10
5,535,375	A	*	7/1996	Eshel et al.	703/27
5,537,592	A		7/1996	King et al.	707/200
5,596,755	A		1/1997	Pletcher et al.	710/261
5,627,996	A		5/1997	Bauer	703/20
5,694,606	A		12/1997	Pletcher et al.	710/261
5,745,752	A		4/1998	Hurvig et al.	707/200
5,745,902	A		4/1998	Miller et al.	707/200
5,754,848	A		5/1998	Hanes	707/200
5,758,352	A		5/1998	Reynolds et al.	707/200
5,761,675	A	*	6/1998	Isenberg	707/200
5,765,169	A		6/1998	Conner	707/200
5,819,275	A		10/1998	Badger et al.	707/100
5,926,805	A		7/1999	Hurvig et al.	707/2
6,055,527	A		4/2000	Badger et al.	707/2

OTHER PUBLICATIONS

Manes, S. "Taking a Gamble with WordVision", PC Magazine, vol. 3, No. 6, pp. 211–221, Apr. 3, 1984.*

Jeffries, R. "What's Ahead for DOS", PC Magazine, vol. 4, No. 24, pp. 95–97, Nov. 26, 1985.*

Prosise, J. "Retrofitting a DOS System", PC Magazine, vol. 5, No. 21, pp. 303–313, Dec. 9, 1986.*

Sanders, S.L., et. "The $R/O Read Only", Jun. 1987.*

Wendin Inc. "Wendin to Release Wendin–DOS in September", PR Newswire Press Release, Aug. 28, 1987.*

Mefford, M.J. "Adding Notes to Directories", PC Magazine, vol. 6, No. 15, pp. 385–394, Sep. 15, 1987.*

Petzold, C. "OS–2: A New Beginning for PC Applications", PC Magazine, vol. 7, No. 7, pp. 273–281, Apr. 12, 1988.*

Digital Research Inc. "Digital Research Introduces POMable, Single–User DOS–Compatible Operating System", Press Release, Jun. 8, 1988.*

Derfler, F.J. Jr. "Building Workgroup Solutions: AppleTalk", PC Magazine, vol. 7, No. 22, pp. 151–160, Dec. 27, 1988.*

World Software Corporation "Extend–A–Name: The 60 Character File Name Utility", User's Guide and Reference Manual, 1988.*

Duncan, R. "Comparing DOS and OS–2 File Systems", PC Magazine, vol. 8, No. 3, pp. 321–327, Feb. 14, 1989.*

Duncan, R. "Design Goals and Implementation of the New High Performance File System", Microsoft Systems Journal, pp. 1–13, Sep. 1989.*

Bonner, P. "What's In a Name?", PC Computing, vol. 2, No. 9, pp. 169–170, Sep. 1989.*

Gralla, P. "Factors Impeding the OS/2 Operating System", PC Computing, vol. 2, No. 12, p. 13, Dec. 1989.*

Greenberg, R.M. "Compress and Expand the Files on Your Hard Disk Automatically", PC Magazine, vol. 8, No. 21, pp. 299–310, Dec. 12, 1989.*

Petzold, C. "1989: The Year in Operating Systems", PC Magazine, vol. 9, No. 1, p. 172, Jan. 16, 1990.*

Duncan, R. "Getting Acquainted with the Latest Version of OS/2: 1.2 (Part 1)", PC Magazine, vol. 9, No. 6, pp. 343–346, Mar. 27, 1990.*

Duncan, R. "Getting Acquainted with the Latest Version of OS/2: 1.2 (Part 2)", PC Magazine, vol. 9, No. 7, pp. 317–321, Apr. 10, 1990.*

Duncan, R. "Using Long Filenames and Extended Attributes (Part 1)", PC Magazine, vol. 9, No. 8, pp. 317–323, Apr. 24, 1990.*

McCormick, J. "Presentation Manager Under OS/2 Encourages Lengthy Name–Calling", Government Computer News, vol. 9, No. 10, pp. 16–17, May 14, 1990.*

Duncan, R. "Using Long Filenames and Extended Attributes (Part 2)", PC Magazine, vol. 9, No. 9, pp. 305–309, May 15, 1990.*

Winship, S. "DOS Shells", PC Week, vol. 7, No. 25, p. 128, Jun. 25, 1990.*

Neuhaus, T. "Databases", PC Magazine, vol. 9, No. 12, pp. 435–437, Jun. 26, 1990.*

Jackson, P. "Apple Talk: The Need for Coexistence Between Apple Macintosh and the IBM PC", PC User, No. 145, p. 171, Nov. 7, 1990.*

Bonner, P. "GeoWorks Ensemble is a One–Size–Fits–All GUI", PC Computing, vol. 3, No. 12, pp. 45–46, Dec. 1990.*

Acerson, K.L. "WordPerfect® 5.1: The Complete Reference", Berkeley:Osborne McGraw–Hill, pp. 272–278, 593–595, 612–615, 1246. ISBN 0–07–881634–3. Z52.5.W65A26 1990.*

"CD–ROM: Rock Ridge Group Submits Preliminary CD–ROM Specs to NIST", EDGE: Work–Group Computing Report, vol. 2, No. 44, Mar. 25, 1991.*

Rock Ridge Technical Working Group Rock Ridge Interchange Protocol Version 1, Rev. 1.09, Jul. 24, 1991.*

Hayes, F. "Making CD–ROM Usable for Unix", Unix World, vol. VIII, No. 7, p. 123, Jul. 1991.*

Somerson, P. "DOS 5.0: Microsoft Corporation's Improved Operating System", PC Computing, vol. 4, No. 7, pp. 97–113, Jul. 1991.*

Rizzo, J. "Disks of a Different Color: Running MSDOS Disks on an Apple Macintosh", MacUser, vol. 7, No. 8, pp. 231–234, Aug. 1991.*

Simon, Barry "What Do You Do To Overcome Disk Disasters", PC Magazine, vol. 10, No. 15, pp. 409–414, Sep. 10, 1991.*

Chin, C. "Using Unused Bytes in Directory Entry?", excerpt from comp.os.msdos.programmer newsgroup thread, Oct. 14, 1991.*

Proteo Technology "Proteo Technology Announces Way You Work New PC Productivity Software That Lets People Master Their PCs Without Knowing DOS", Press Release, Oct. 21, 1991.*

Young, A. "The CD–ROM Standards Frontier: Rock Ridge", CD–ROM Professional, vol. 4, No. 6, pp. 53–56, Nov. 1991.*

Zelnick, N. "Way You Work Does It Your Way", PC Magazine, vol. 10, No. 22, pp. 62–63, Dec. 31, 1991.*

Bonner, P. "Build a Document Manager Under Windows", PC Computing, vol. 4, No. 12, pp. 275–281, Dec. 1991.*

Hotch, R. "Will This Be The Way You Work?", Nation's Business, Mar. 1992.*

Prosise, J. "Tutor", PC Magazine, vol. 11, No. 5, pp. 397–399, Mar. 17, 1992.*
Bonner, P. "Windows 3.1: Faster and Smarter", PC Computing, vol. 5, No. 5, pp. 128–139, May 1992.*
Smith, G. "OS/2 2.0 Does the Job", PC Computing, vol. 5, No. 5, pp. 48–55, May 1992.*
Busch, D.D. "4DOS: DOS As You Like It", Computer Shopper, vol. 12, No. 7, pp. 698–699, Jul. 1992.*
Gralla, P. "Shareware", Computer Shopper, vol. 12, No. 9, pp. 701–703, Sep. 1992.*
Somerson, P. "Spy–Proof Your PC: 13 Ingenious Ways to Keep Your System Secure", PC Computing, vol. 5, No. 9, pp. 218–237, Sep. 1992.*
Ruley, J.D. "Feature–Rich Beta at a Bargain Price", Windows Magazine, Oct. 1, 1992.*
Giovetti, A.C. "Way You Work: Personal Office", Compute!, Issue 146, p. 126, Nov. 1992.*
Rohan, R. "Golden Retriever Fetches Files in Windows", Computer Shopper, vol. 12, No. 11, p. 947, Nov. 1992.*
Microsoft Chicago Long Filename Specification, Revision 0.5, Dec. 4, 1992.*
Goh, S. "MS DOS 6.0", excerpt of a comp.os.msdos.misc newsgroup thread, Feb. 20, 1993.*
"Access Review Too Harsh", Letters Column, Windows Magazine, No. 403, p. 18, Mar. 1, 1993.*
Maird, M.K. "What are the Benefits of 4DOS?", excerpt of a comp.os.msdos.4dos newsgroup thread, Mar. 7, 1993.*
Above Software, "Above Software Introduces Golden Retriever 2.0b", Press Release, Mar. 29, 1993.*
O'Malley, C. "Fetching Desktop Files: Standalone Document Managers", Windows Sources, vol. 1, No. 2, p. 443, Mar. 1993.*
Fowler, D. "Cross Talking: Sharing Files Over a Mixed–Platform Network", Computer Shopper, vol. 13, No. 3, pp. 783–785, Mar. 1993.*
Berst, J. "Come Closer and I'll Tell You Some Secrets About Windows 4.0", Windows Magazine, No. 404, p. 43, Apr. 1, 1993.*
Mallory, J. "Breakthrough on DOS Filename Limits", Newsbytes, Apr. 12, 1993.*
Almax Software "Longer Filenames for DOS", Press Release, May 1993.*
Rettig, H. "Custom Windows Made Easy", Windows Magazine, No. 406, p. 264, Jun. 1, 1993.*
Capen, T. "The Ultimate File Manager", Corporate Computing, vol. 2, No. 6, p. 54, Jun. 1993.*
Lincoln, S. "Death to 8+3 Filenames!", excerpt from a comp.os.ms–windows.apps newsgroup thread, Jul. 9, 1993.*
Lewallen, D., F. Scot and E. Bott "The NT Desktop—Like Windows, Only Better", PC Computing, vol. 6, No. 7, pp. 124–127, Jul. 1993.*
Nilsson, B.A. "Sherlock Solves the Case of the Cryptic Windows Filenames", Computer Shopper, vol. 13, No. 7, p. 434, Jul. 1993.*
2010 Software "Sherlock 2.0 Released by 2010 Software", Press Release, Aug. 9, 1993.*
Clark, I.M. "Proposed DOS Header Record", excerpt from comp.os.msdos.programmer newsgroup thread, Aug. 11, 1993.*
Allen, J. et al. "The Vices and Virtues of MS–DOS", PC Computing, vol. 6, No. 8, pp. 27–31, Aug. 1993.*
Wagner, M. "Developers to Get Tour of 'Chicago'—New Microsoft OS", Open Systems Today, No. 136, p. 3, Nov. 8, 1993.*
Idol, C. "Sherlock", Compute, vol. 15, No. 11, p. 150, Nov. 1993.*
DeVoney, C. and R.C. Kennedy "NT Has Arrived", Windows Sources, vol. 1, No. 10, pp. 283–294, Nov. 1993.*
Davis, F.E. "NT, No Thanks; Wait for 4.0", Windows Sources, vol. 1, No. 10, pp. 105–106, Nov. 1993.*
Mathisen, T. "Novell's DOS 7 Offers Multitasking, Memory Management, and Peer–to–Peer Networking. Is It a Better DOS Than Microsoft's?", Byte Magazine, Jun. 1994.*
Olsen, M. "Student Writes Free Version of DOS", University of Wisconsin at River Falls Student Voice, Dec. 1, 1994.*
Saiedian, H. and M. Siddiqi "A Framework for the Assessment of Operating Systems for Small Computers", ACM SIGICE Bulletin, vol. 21, No. 4, pp. 2–27, Apr. 1996.*
Cluts, N.W. "Making Room for Long Filenames", Microsoft Developer Network Technology Team, downloaded from msdn.microsoft.com/library/en–us/dnfiles/html/msdn_longfile.asp?frame=true, Aug. 1996.*
Styer, E. "Disks", ACM SIGICE Bulletin, vol. 23, No. 1, pp. 22–32, Jul. 1997.*
vinDaci "Long Filename Specification", downloaded from home.teleport.com/~brainy/lfn.htm, Jan. 6, 1998.*
Mattias "25 Years of DR DOS History", downloaded from freedos.sourceforge.net/freedos/news/bits/drdos–hist.txt, Sep. 18, 2000.*
The PC Guide "File Allocation Tables", downloaded from www.pcguide.com/ref/hdd/file, Apr. 17, 2001.*
Tanenbaum, A.S. "Modern Operating Systems", Upper Saddle River:Prentice Hall, pp. 430–447, 2001.*
Hall, J. "An Overview of FreeDOS", downloaded from freedos.sourceforge.net/freedos/news/bits/article_fsm.txt, 2002.*
Beta Systems "DOS History", downloaded from www.pow-ernet.co.za/info/DOS/His.Htm, Apr. 30, 2003.*
Fuchs, C. "DOS Frequently Asked Questions", downloaded from www.drdos.new/faq, Jan. 1, 2004.*
Jh "20 Years of DOS History", downloaded from freedos.sourceforge,net/freedos/news/bits/doshist.txt, Mar. 25, 2005.*
Wikipedia "Comparison of File Systems", downloaded from www.wikipedia.org, Aug. 29, 2005.*
Wikipedia "DOS", downloaded from www.wikipedia.org, Sep. 15, 2005.*
Williams, D. "DOS Technical Information: Programming Technical Reference", downloaded from http://www.freein-fosociety.com/computers/dostechnical.html, 1988.*
Bonner, Paul, "What's in a Name?," Sep. 1988, *PC–Computing,* vol. 2,(9), p. 169(2).
Bonner, Paul, "Build a Document Manager Under Windows," Dec. 1991, *PC–Computing,* vol. 4(12), p. 275–277, 280–283.
Comer, D. et al., "The Tilde File Naming Scheme" *IEEE 6th International Conference on Distributed Computing Systems,* Cambridge, Massachusetts, May 23, 1986, pp. 509–514.
William M. Crow, "Encapsulation of Applications in the NewWave Environment," *Hewlett–Packard Journal,* Aug. 1989, 40(4), p. 57–64.
Ray Duncan, "Design Goals and Implementation of the New High Performance File System" *Microsoft Systems Journal,* Sep. 1989, 4(5), pp. 1–13.

Ray Duncan, "Using Long Filenames and Extended Attributes" *Parts 1 & 2, PC Magazine,* Apr. 24 & May 15, 1990, 9(8,9), pp. 317–323 & 305–309.
Les Freed, "High–End PC–to–MAC LAN Solutions", *PC Magazine,* May 1992, 11(9), p. 203(8).
Glass, Brett, "Create Your Own Environment," *PC–Computing,* Oct. 1990, 3(10), 106–111.
Hurwicz, Mike, "MS–DOS 3.1 Makes It Easy to Use IBM PCs on a Network," *Data Communications,* Nov. 1985, 223–237.
Mallory, Jim, "Breakthrough on DOS Filename Limits," *Newsbytes News Network,* Apr. 12, 1993.
McCormick, John, "Presentation Manager Under OS/2 Encourages Lengthy Name–Calling," *Government Computer News,* May 14, 1990, 9(10), p. 16(2).
O'Malley, Chris, "Fetching Desktop Files: Standalone Document Managers," *Window Sources,* Mar. 1993, 1(2), p. 443(1).
Rohan, Rebecca, "Golden Retriever Fetches Files in Windows," *Computer Shopper,* Nov. 1992, 12(11), p. 947(1).
Samuel J. Leffler et al., "The Design and Implementation of the 4.3 BSD UNIX Operating System," *Addison–Wesley Publishing Company* 1989, Chapter 2, pp. 34–36.
Trivette, Donald B., "Utility Provides 60–Character Filenames," *PC Magazine,* Sep. 1988, 7(16), p. 56(1).
Wang, Y.E.G., "Universal File Names for Ada," *Ada Letters,* Jan./Feb. 1990, Integrated Software, Inc., New York, NY, vol. X(1), pp. 111–117.
"The Intelligent Way to Search," Oct. 1987, News Release, Dateline: Burlington, MA.
"File sharing Protocol," Microsoft Corporation, Nov. 7, 1988, 71 pages.
"World Software Corporation (WSC) Launches Extend–A–Name in Europe," *Computer Product Update,* Jul. 27, 1990.
"Above Software Introduces Golden Retriever 2.0b'," News Release, Dateline: Irvine, CA, Mar. 29, 1993.
Len, A.F. et al., "New Improved Windows," *PC World,* Dec. 1993, 11(12), p. 252(3).
Mark G. Sobell , "A Practical Guide to the Unix System," Dec. 1989, System V Release 3 and BSD 4.3, pp. 12–14, 32, 66–69, 82–83 and 88–89.
"Appendix C How Filenames Are Converted," *Microsoft LAN Manager Services for Macintosh Administrator's Guide,* Jun. 1991, Version 1.0, for Microsoft Operating System/2, Microsoft Corporation, pp. 119–123.
"Long Filename Specification", Hardware White Paper, Designing Hardware for Microsoft® Operating Systems, Microsoft Corporation, Dec. 4, 1992, Version 0.5, http://www.osdever.net/documents/longfilename.pdf?the id=39, 19 pages.
Rock Ridge Interchange Protocol, Version 1, An ISO 9660:1998 Compliant Approach to Providing Adequate CD–Rom Support for Posix File System Semantics; *Rock Ridge Technical Working Group,* Revision 1.09, Jul. 24, 1991, 39 pages.
"Rock Ridge Group Submits Preliminary CD–ROM Specs to NIST—Sixteen Companies Offer Their Support", *The Florida SunFlash,* Mar. 1991, 27(11), 3 pages.
Translation into English of Plaintiff's Grounds for Nullity, *Dr. Friedrich–Karl Boese* v. *Microsoft Corporation,* Dec. 21, 2004, 1–25.
Merkmalsanalyse Anspruch 1, Exhibit 5 of Reference 76, 1 page.
Merkmalsanalyse Anspruch 12, Exhibit 6 of Reference 76, 1 page.
Article from Newsgroup comp.archives, Exhibit 7a of Reference 76, Aug. 21, 1992, 1 page.
Article from Newsgroup comp.std.misc, Exhibit 7b of Reference 76, Mar. 21, 1991, 3 pages.
Article from Newsgroup comp.new.prod, Exhibit 7c of Reference 76, Aug. 20, 1991, 1 page.
Article from Newsgroup comp.unix.bsd and comp.os.linux, Exhibit 7d of Reference 76, Dec. 12, 1992, 1 page.
Response to Official Communication re: EP Application No. 94 105 169.0–2201 dated May 10, 2000, Exhibit 14 of Reference 76, 9 pages.
"Common Name Space for Long and Short Filenames," *Microsoft Corporation,* Exhibit 15 of Reference 76, 34 pages.
Communication Under Rule 51(4) EPC re: EP Application No. 94 105 169.0–2201, Dec. 19, 2000, Exhibit 16 of Reference 76, 39 pages.
Rock Ridge Interchange Protocol, Version 1, An ISO 9660:1988 Compliant Approach to Providing Adequate CD–Rom Support for POSIX File System Semantics; Rock Ridge Technical Working Group, Revision 1.09, Jul. 24, 1991.
"Rock Ridge Group Submits Preliminary CD–ROM Specs to NIST—Sixteen Companies Offer Their Support" *The Florida SunFlash,* vol. 27 #11, Mar. 1991.

* cited by examiner

EX PARTE REEXAMINATION CERTIFICATE ISSUED UNDER 35 U.S.C. 307

THE PATENT IS HEREBY AMENDED AS INDICATED BELOW.

Matter enclosed in heavy brackets [ ] appeared in the patent, but has been deleted and is no longer a part of the patent; matter printed in italics indicates additions made to the patent.

AS A RESULT OF REEXAMINATION, IT HAS BEEN DETERMINED THAT:

The patentability of claims **2–4** is confirmed.

Claim **1** is determined to be patentable as amended.

New claims **5–44** are added and determined to be patentable.

1. In a computer system having a processor running an operating system and a memory means storing the operating system, a method comprising the computer-implemented steps of:

(a) storing in the memory means a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters that is permissible by the operating system;

(b) storing in the memory means a second directory entry for **[**a**]** the file wherein the second directory entry holds a long filename for the file and wherein the second directory entry includes an attributes field **[**which may be set to make the second directory entry invisible to the operating system**]** and the step of storing the second directory entry further comprises the step of setting the attributes field so that the second directory entry is invisible to **[**the**]** *a second* operating system, said long filename including more than the maximum number of characters that is permissible by the operating system; and

(c) accessing the first directory entry with the operating system.

5. The method of claim 1, wherein the attributes field comprises a volume label bit and wherein the step of setting the attributes field so that the second directory entry is invisible to the second operating system comprises setting the volume label bit to a value of one.

6. The method of claim 5, wherein the attributes field further comprises a read-only bit, a hidden bit, and a system bit, and wherein the step of setting the attributes field so that the second directory entry is invisible to the second operating system further comprises setting each of the read-only bit, the hidden bit, and the system bit to a value of one.

7. The method recited in claim 1, wherein the first and second directory entries for the file each have a beginning cluster field, and wherein the method further comprises:

storing, in the beginning cluster field of the first directory entry, a pointer to a section of the memory means where a first cluster of the file is stored, and

storing, in the beginning cluster field of the second directory entry, a value of zero.

8. The method of claim 2, wherein the second directory entry includes an attributes field comprising a plurality of bits, and wherein the method further comprises the step of setting the bits of the attributes field so that the second directory entry is invisible to a second operating system that supports only short filenames.

9. The method of claim 8, wherein one of the bits of the attributes field comprises a volume label bit, and wherein the step of setting the attributes field so that the second directory entry is invisible to the second operating system comprises setting the volume label bit to a value of one.

10. The method of claim 9, wherein the bits of the attributes field further comprise a hidden bit, a read-only bit, and a system bit, and wherein the step of setting the attributes field so that the second directory entry is invisible to the second operating system comprises setting each of the hidden bit, read-only bit and system bit to a value of one.

11. The method recited in claim 2, wherein the first and second directory entries for the file each have a beginning cluster field, and wherein the method further comprises:

storing, in the beginning cluster field of the first directory entry, a pointer to a section of the memory means where a first cluster of the file is stored; and

storing, in the beginning cluster field of the second directory entry, a value of zero.

12. The method of claim 3, wherein the second directory entry and the at least one additional directory entry each includes an attributes field comprising a plurality of bits, and wherein the method further comprises the step of setting the bits of the attributes fields of the second directory entry and the at least one additional directory entry so that those directory entries are invisible to a second operating system that supports only short filenames.

13. The method of claim 12, wherein one of the bits of the attributes field of each of the second directory entry and the at least one additional directory entry comprises a volume label bit, and wherein the step of setting the attributes fields of those entries so that those entries are invisible to the second operating system comprises setting the volume label bit a value of one.

14. The method of claim 13, wherein the bits of the attributes field of each of the second directory entry and the at least one additional directory entry further comprise a hidden bit, a read-only bit, and a system bit, and wherein the step of setting the attributes fields of those entries so that those entries are invisible to the second operating system comprises setting each of the hidden bit, read-only bit and system bit to a value of one.

15. The method recited in claim 3, wherein the first directory entry, the second directory entry, and the at least one additional directory entry for the file each have a beginning cluster field, and wherein the method further comprises:

storing, in the beginning cluster field of the first directory entry, a pointer to a section of the memory means where a first cluster of the file is stored; and

storing, in the beginning cluster field of each of the second directory entry and the at least one additional directory entry, a value of zero.

16. The method of claim 4, wherein the second directory entry and the at least one additional directory entry each includes an attributes field comprising a plurality of bits, and wherein the method further comprises the step of setting the bits of the attributes fields of the second directory entry and the at least one additional directory entry so that those directory entries are invisible to a second operating system that supports only short filenames.

17. The method of claim 16, wherein one of the bits of the attributes field of each of the second directory entry and the

at least one additional directory entry comprises a volume label bit, and wherein the step of setting the attributes fields of those entries so that those entries are invisible to the second operating system comprises setting the volume label bit to a value of one.

18. The method of claim 17, wherein the bits of the attributes field of each of the second directory entry and the at least one additional directory entry further comprise a hidden bit, a read-only bit, and a system bit, and wherein the step of setting the attributes fields of those entries so that those entries are invisible to the second operating system comprises setting each of the hidden bit, read-only bit and system bit to a value of one.

19. The method recited in claim 4, wherein the first directory entry, the second directory entry, and the at least one additional directory entry for the file each have a beginning cluster field, and wherein the method further comprises:

storing, in the beginning cluster field of the first directory entry, a pointer to a section of the memory means where a first cluster of the tile is stored; and

storing, in the beginning cluster field of each of the second directory entry and the at least one additional directory entry, a value of zero.

20. The method of claim 4, wherein the signature comprises characters that are not permitted to be used as characters of a short filename of a file.

21. The method of claim 4, further comprising storing a checksum of the short filename in the second directory entry and in the at least one additional directory entry.

22. In a computer system having a processor running an operating system and a memory means storing the operating system, a method comprising the computer-implemented steps of:

(a) storing in the memory means, as part of a directory structure of a file system of the operating system, a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters permitted for short filenames;

(b) storing in the memory means, as part of the same directory structure, a second directory entry for the file wherein the second directory entry holds a long filename for the file, said long filename including more than the maximum number of characters that is permissible for short filenames, and wherein the second directory entry includes an attributes field comprising a plurality of bits and the step of storing the second directory entry further comprises the step of setting the bits of the attributes field so that the second directory entry is invisible to a second operating system that only supports short filenames; and

(c) accessing the first directory entry with the operating system.

23. The method of claim 22, wherein the attributes field comprises a volume label bit and wherein the step of setting the attributes field so that the second directory entry is invisible to the second operating system comprises setting the volume label bit to a value of one.

24. The method of claim 23, wherein the attributes field further comprises a read-only bit, a hidden bit, and a system bit, and wherein the step of setting the attributes field so that the second directory entry is invisible to the second operating system further comprises setting each of the read-only bit, the hidden bit, and the system bit to a value of one.

25. The method recited in claim 22, wherein the first and second directory entries for the file each have a beginning cluster field, and wherein the method further comprises:

storing, in the beginning cluster field of the first directory entry, a pointer to a section of the memory means where a first cluster of the file is stored; and

storing, in the beginning cluster field of the second directory entry, a value of zero.

26. In a computer system having a processor running an operating system and a memory means storing the operating system, a method, comprising the computer-implemented steps of:

(a) storing in the memory means, as part of a directory structure of a file system of the operating system, a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters permitted for short filenames;

(b) storing in the memory means, as part of the same directory structure, a second directory entry for the file wherein the second directory entry holds a long filename for the file, said long filename including more than the maximum number of characters that is permissible for short filenames;

(c) storing a checksum of the short filename in the second directory entry; and

(d) accessing the first directory entry with the operating system.

27. The method of claim 26, wherein the second directory entry includes an attributes field comprising a plurality of bits, and wherein the method further comprises the step of setting the bits of the attributes field so that the second directory entry is invisible to a second operating system that supports only short filenames.

28. The method of claim 27, wherein one of the bits of the attributes field comprises a volume label bit, and wherein the step of setting the attributes field so that the second directory entry is invisible to the second operating system comprises setting the volume label bit to a value of one.

29. The method of claim 28, wherein the bits of the attributes field further comprise a hidden bit, a read-only bit, and a system bit, and wherein the step of setting the attributes field so that the second directory entry is invisible to the second operating system comprises setting each of the hidden bit, read-only bit and system bit to a value of one.

30. The method recited in claim 26, wherein the first and second directory entries for the file each have a beginning cluster field, and wherein the method further comprises:

storing, in the beginning cluster field of the first directory entry, a pointer to a section of the memory means where a first cluster of the file is stored, and

storing, in the beginning cluster field of the second directory entry, a value of zero.

31. In a computer system having a processor running an operating system and a memory means storing the operating system, a method, comprising the computer-implemented steps of:

(a) storing in the memory means, as part of a directory structure of a file system of the operating system, a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters permitted for short filenames;

(b) storing in the memory means, as part of the same directory structure, a second directory entry for the file wherein the second directory entry holds a long filename for the file, said long filename including more than the maximum number of characters that is permissible for short filenames;

(c) storing in the memory means, as part of the same directory structure, at least one additional directory entry holding a next portion of the long filename;

(d) storing in each of the second directory entry and the at least one additional directory entry a checksum of the short filename; and

(e) accessing the first directory entry with the operating system.

32. The method of claim 31, wherein the second directory entry and the at least one additional directory entry each includes an attributes field comprising a plurality of bits, and wherein the method further comprises the step of setting the bits of the attributes fields of the second directory entry and the at least one additional directory entry so that those directory entries are invisible to a second operating system that supports only short filenames.

33. The method of claim 32, wherein one of the bits of the attributes field of each of the second directory entry and the at least one additional directory entry comprises a volume label bit, and wherein the step of setting the attributes fields of those entries so that those entries are invisible to the second operating system comprises setting the volume label bit to a value of one.

34. The method of claim 33, wherein the bits of the attributes field of each of the second directory entry and the at least one additional directory entry further comprise a hidden bit, a read-only bit, and a system bit, and wherein the step of setting the attributes fields of those entries so that those entries are invisible to the second operating system comprises setting each of the hidden bit, read-only bit and system bit to a value of one.

35. The method recited in claim 31, wherein the first directory entry the second directory entry, and the at least one additional directory entry for the file each have a beginning cluster field, and wherein the method further comprises:

storing, in the beginning cluster field of the first directory entry, a pointer to a section of the memory means where a first cluster of the file is stored; and

storing, in the beginning cluster field of each of the second directory entry and the at least one additional directory entry, a value of zero.

36. In a computer system having a processor running an operating system and a memory means storing the operating system, a method, comprising the computer-implemented steps of:

(a) storing in the memory means, as part of a directory structure of a file system of the operating system, a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters permitted for short filenames;

(b) storing in the memory means, as part of the same directory structure, a second directory entry for the file wherein the second directory entry holds a long filename for the file, said long filename including more than the maximum number of characters that is permissible for short filenames;

(c) accessing the first directory entry with the operating system;

(d) storing in the memory means, as part of the same directory structure, at least one additional directory entry holding a next portion of the long filename and a signature that uniquely identifies which portion of the long filename.

37. The method of claim 36, wherein the second directory entry and the at least one additional directory entry each includes an attributes field comprising a plurality of bits, and wherein the method further comprises the step of setting the bits of the attributes fields of the second directory entry and the at least one additional directory entry so that those directory entries are invisible to a second operating system that supports only short filenames.

38. The method of claim 37, wherein one of the bits of the attributes field of each of the second directory entry and the at least one additional directory entry comprises a volume label bit, and wherein the step of setting the attributes fields of those entries so that those entries are invisible to the second operating system comprises setting the volume label bit to a value of one.

39. The method of claim 38, wherein the bits of the attributes field of each of the second directory entry and the at least one additional directory entry further comprise a hidden bit, a read-only bit, and a system bit, and wherein the step of setting the attributes fields of those entries so that those entries are invisible to the second operating system comprises setting each of the hidden bit, read-only bit and system bit to a value of one.

40. The method recited in claim 36, wherein the first directory entry, the second directory entry, and the at least one additional directory entry for the file each have a beginning cluster field, and wherein the method further comprises:

storing, in the beginning cluster field of the first directory entry, a pointer to a section of the memory means where a first cluster of the file is stored; and

storing, in the beginning cluster field of each of the second directory entry and the at least one additional directory entry, a value of zero.

41. The method of claim 36, wherein the signature comprises characters that are not permitted to be used as characters of a short filename of a file.

42. The method of claim 36, further comprising storing a checksum of the short filename in the second directory entry and in the at least one additional directory entry.

43. The method of claim 36, wherein the second directory entry immediately follows the first directory entry in the directory structure.

44. The method of claim 43, wherein the at least one additional directory entry immediately follows the second directory entry in the directory structure.

* * * * *

US005758352A

United States Patent [19]

Reynolds et al.

[11] Patent Number: 5,758,352

[45] Date of Patent: May 26, 1998

[54] **COMMON NAME SPACE FOR LONG AND SHORT FILENAMES**

[75] Inventors: **Aaron R. Reynolds**, Redmond; **Dennis R. Adler**, Mercer Island; **Ralph A. Lipe**, Woodinville; **Ray D. Pedrizetti**, Issaquah; **Jeffrey T. Parsons**; **Rasipuram V. Arun**, both of Redmond, all of Wash.

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

[21] Appl. No.: **711,692**

[22] Filed: **Sep. 5, 1996**

Related U.S. Application Data

[63] Continuation of Ser. No. 427,004, Apr. 24, 1995, Pat. No. 5,579,517, which is a continuation of Ser. No. 41,497, Apr. 1, 1993, abandoned.

[51] **Int. Cl.**[6] **G06F 17/30**

[52] **U.S. Cl.** **707/200**; 707/1; 707/6

[58] **Field of Search** 395/600; 364/200

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,780,821	10/1988	Crossley .	
4,987,531	1/1991	Nishikado et al.	364/200
5,307,494	4/1994	Yasumatsu et al.	395/600
5,313,646	5/1994	Hendricks et al.	395/600
5,359,725	10/1994	Garcia et al.	395/500
5,363,487	11/1994	Willman et al. .	
5,371,885	12/1994	Letwin	395/600
5,388,257	2/1995	Bauer	395/600
5,392,427	2/1995	Barrett et al. .	
5,412,808	5/1995	Bauer	395/600
5,421,001	5/1995	Methe	395/500
5,434,974	7/1995	Loucks et al. .	
5,437,029	7/1995	Sinha .	
5,483,652	1/1996	Sudama et al. .	
5,535,375	7/1996	Eshel et al. .	

FOREIGN PATENT DOCUMENTS

1041039	2/1989	Japan	G06F 12/00
1315843	12/1989	Japan	G06F 12/00
2148341	7/1990	Japan	G06F 12/00
6019763	1/1994	Japan	G06F 12/00

OTHER PUBLICATIONS

Hurwicz, Mike, "MS-DOS 3.1 Makes It Easy to Use IBM PCs on a Network." *Data Communications*, Nov., 1985.

"The Intelligent Way to Search," News Release, Dateline: Burlington, MA, Oct., 1987.

Trivette, Donald B., "Utility Provides 60-Character Filenames," *PC Magazine*, vol. 7, N. 16, p.56(1), Sep., 1988.

"File sharing Protocol," Microsoft Corporation, Nov. 7, 1988.

Bonner, Paul, "What's in a Name?," *PC-Computing*, vol. 2, N. 9, p. 169(2), Sep., 1988.

Duncan, Ray, "Power Programming Using Long Filenames and Extended Attributes, Part 1," *PC Magazine*, pp. 317-323, Apr. 24, 1990.

McCormick, John, "Presentation Manager Under OS/2 Encourages Lengthy Name-Calling," *Government Computer News*, vol. 9, N. 10, p. 16(2), May 14, 1990.

Duncan Ray, "Power Programming Using Long Filenames and Extended Attributes, Part 2," *PC Magazine*, pp. 305-309, May 15, 1990.

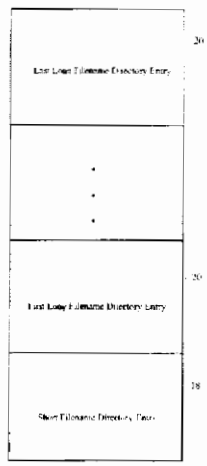
(List continued on next page.)

Primary Examiner—Thomas G. Black
Assistant Examiner—Cheryl R. Lewis
Attorney, Agent, or Firm—Christensen O'Connor Johnson & Kindness PLLC

[57] **ABSTRACT**

An operating system provides a common name space for both long filenames and short filenames. In this common namespace, a long filename and a short filename are provided for each file. Each file has a short filename directory entry and may have at least one long filename directory entry associated with it. The number of long filename directory entries that are associated with a file depends on the number of characters in the long filename of the file. The long filename directory entries are configured to minimize compatibility problems with existing installed program bases.

28 Claims, 8 Drawing Sheets



OTHER PUBLICATIONS

"World Software Corporation (WSC) Launches Extend–A–Name in Europe." *Computer Product Update*, Jul. 27, 1990.

Glass, Brett. "Create Your Own Environment." *PC–Computing*, vol. 3, N. 10, p. 106(6), Oct., 1990.

Bonner, Paul. "Build a Document Manager Under Window." *PC–Computing*, vol. 4, N. 12, p. 275(7), Dec., 1991.

Rohan, Rebecca. "Golden Retriever Fetches Files in Windows." *Computer Shopper*, vol. 12, N. 11, p. 947(1), Nov., 1992.

O'Malley, Chris. "Fetching Desktop Files: Standalone Document Managers." *Window Sources*, vol. 1, N. 2, p. 443(1), Mar., 1993.

"Above Software Introduces Golden Retriever 2.0b'." News Release, Dateline: Irvine, CA, Mar. 29, 1993.

"Breakthrough on DOS File Name Limits." *Newsbytes News Network*, Apr. 12, 1993.

Mallory, Jim. "Breakthrough on DOS Filename Limits." *Newsbytes*, Apr. 12, 1993.

"New Improved Windows." *PC World*, vol. 11, N. 12, p. 252(3), Dec., 1993.

Samuel J. Leffler et al., "The Design and Implementation of the 4.3BSD UNIX Operating System." Addison–Wesley Publishing Company, 1989, Chapter 2, pp. 34–36.

Duncan, Ray. "Design Goals and Implementation of the New High Performance File System." Sep., 1989, pp. 1–13.

Duncan, Ray. "Using Long Filenames and Extended Attributes." PC Magazine, Parts 1 and 2, Apr. 24 and May 15, 1990, vol. 9, Nos. 8 and 9, pp. 317 and 305.

Wang, Y.E.G. "Universal File Names for Ada." Ada Letters, Integrated Software, Inc., New York, NY, Jan./Feb., 1990, pp. 111–117.

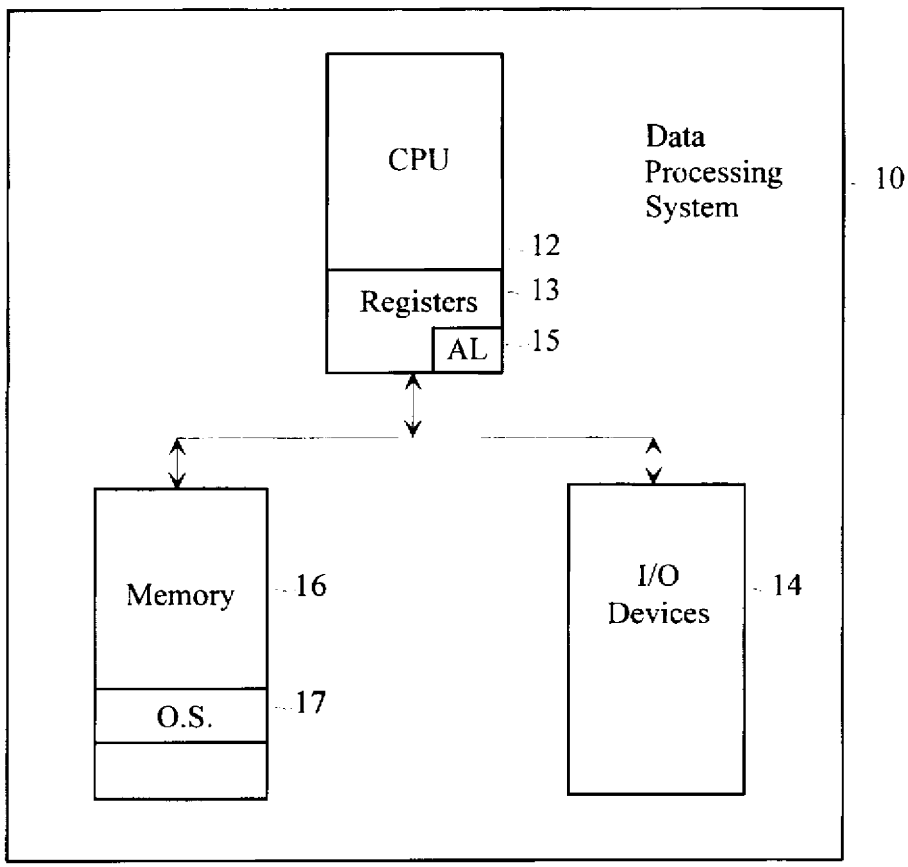

CPU
Data
Processing
System
10
12
Registers
13
AL
15
Memory
16
O.S.
17
I/O
Devices
14

Figure 1

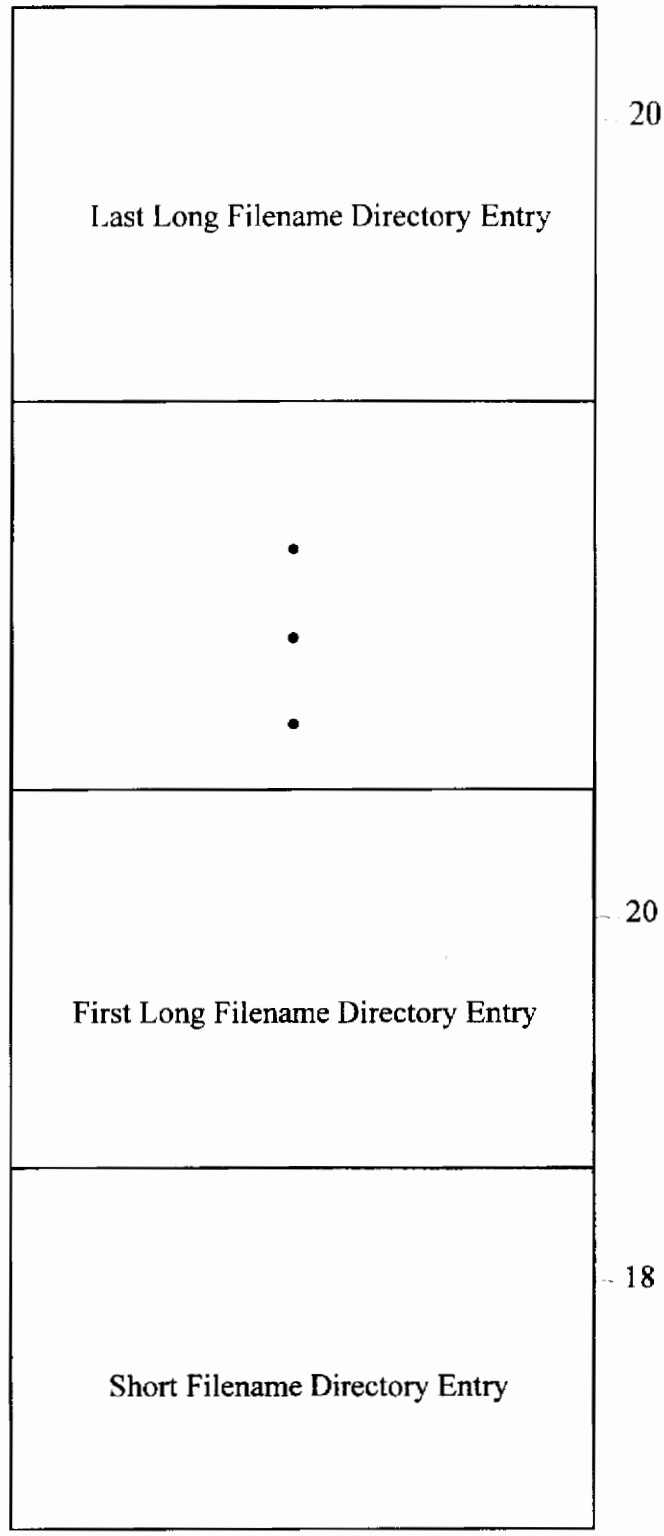

Last Long Filename Directory Entry
20
First Long Filename Directory Entry
20
Short Filename Directory Entry
18

Figure 2

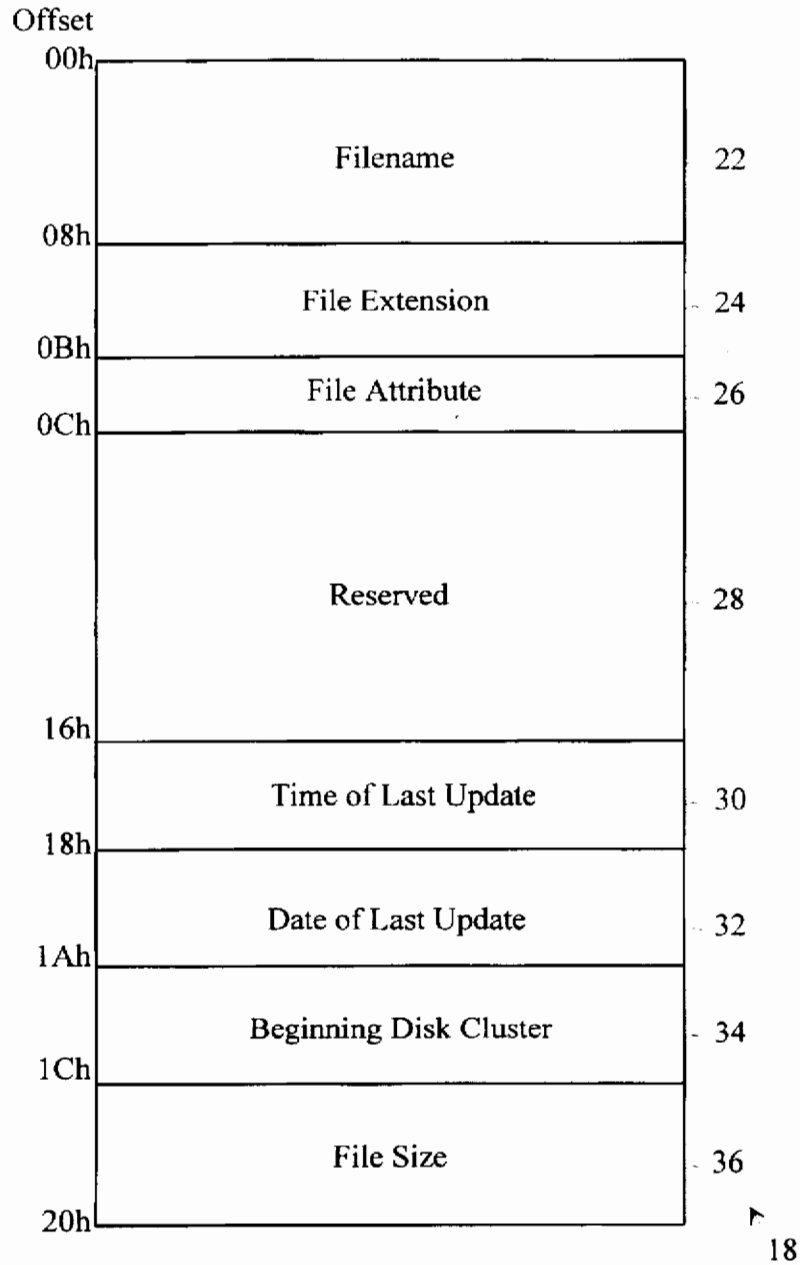

Offset
00h
Filename
22
08h
File Extension
24
0Bh
File Attribute
26
0Ch
Reserved
28
16h
Time of Last Update
30
18h
Date of Last Update
32
1Ah
Beginning Disk Cluster
34
1Ch
File Size
36
20h
18

Figure 3a

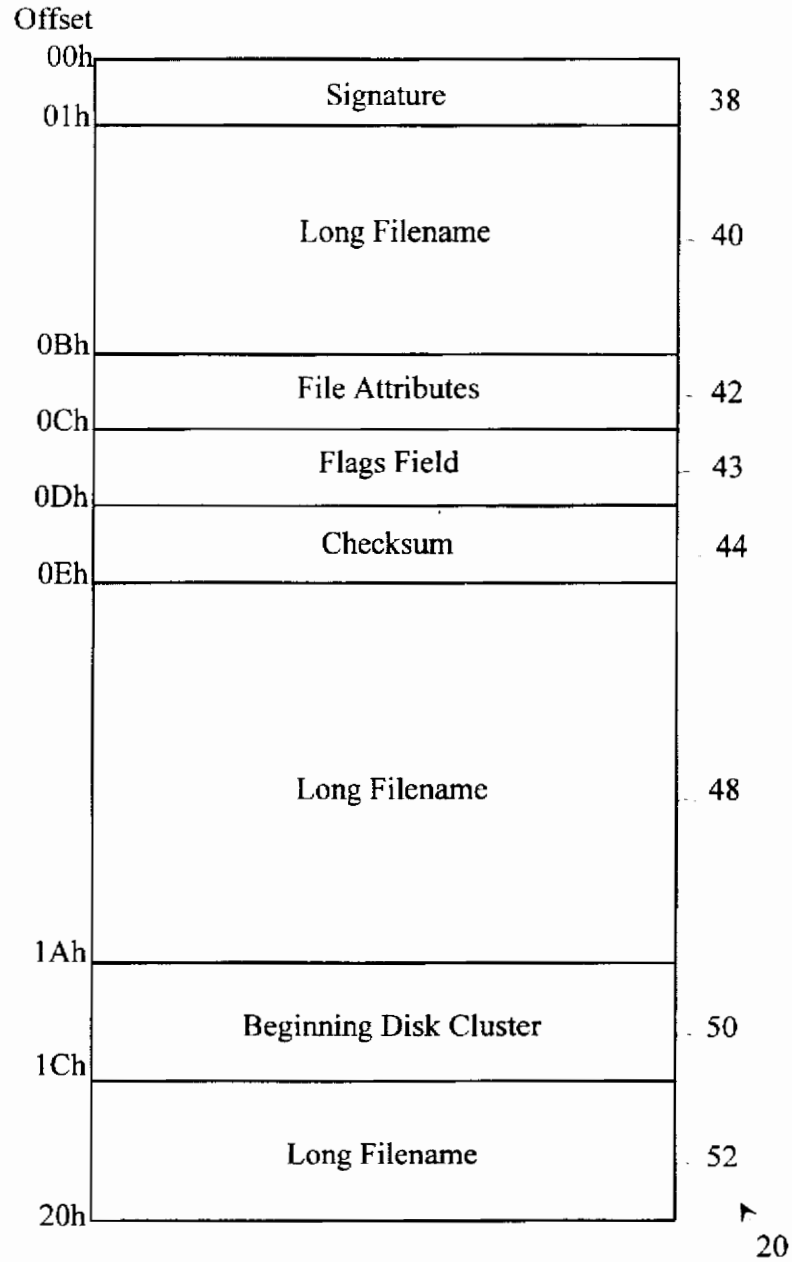

Offset
00h
01h
Signature
38
Long Filename
40
0Bh
File Attributes
42
0Ch
Flags Field
43
0Dh
Checksum
44
0Eh
Long Filename
48
1Ah
Beginning Disk Cluster
50
1Ch
Long Filename
52
20h
20

Figure 3b

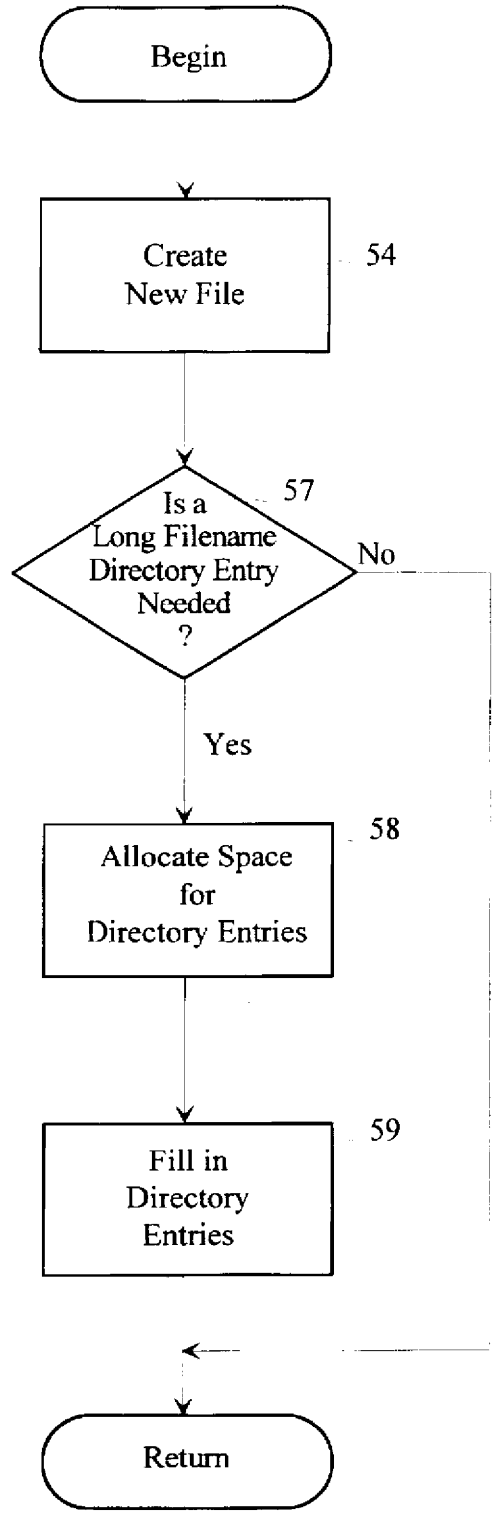

Begin
Create New File
54
Is a Long Filename Directory Entry Needed ?
57
No
Yes
Allocate Space for Directory Entries
58
Fill in Directory Entries
59
Return

Figure 4

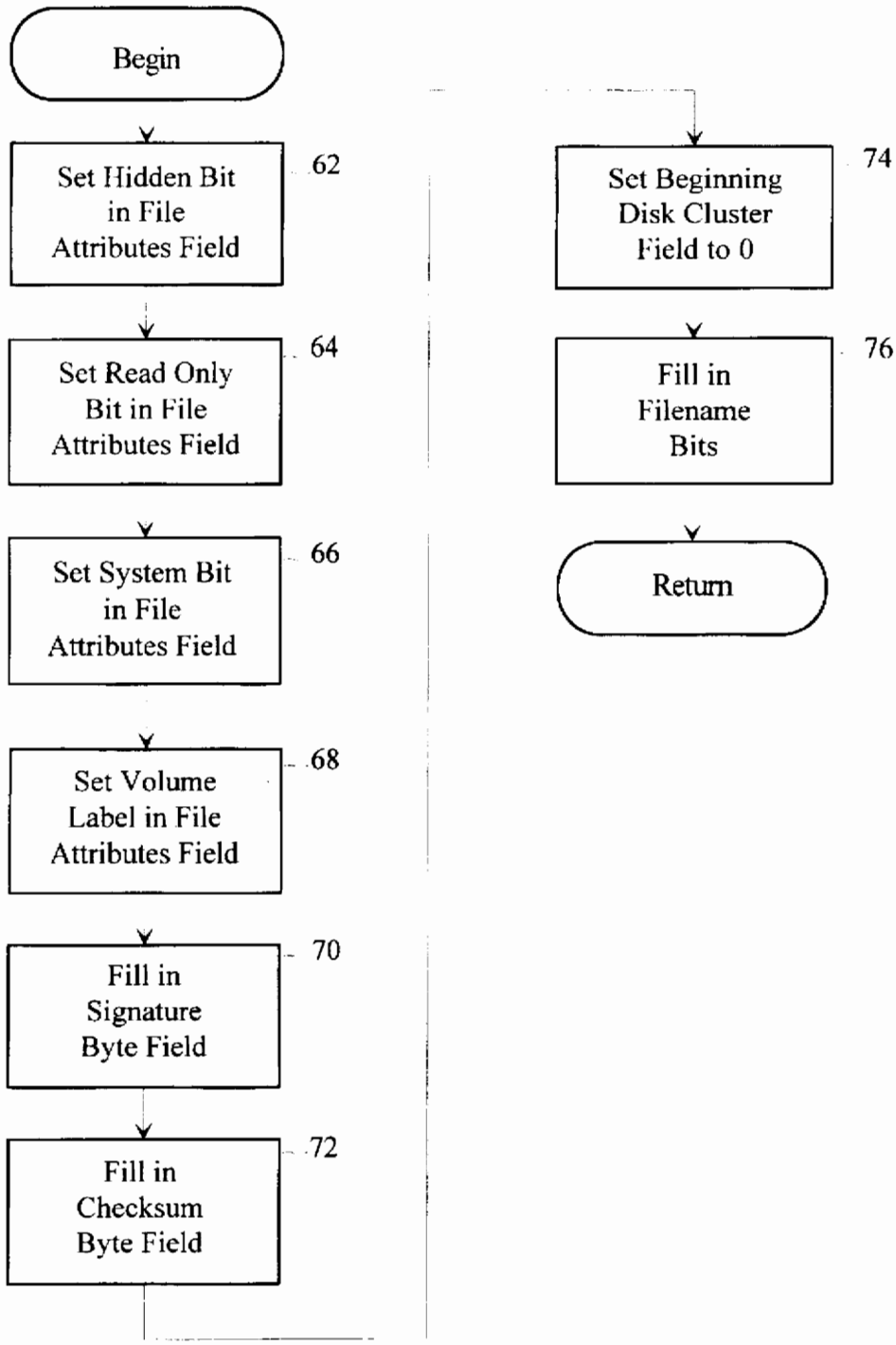

Begin
Set Hidden Bit in File Attributes Field
62
Set Read Only Bit in File Attributes Field
64
Set System Bit in File Attributes Field
66
Set Volume Label in File Attributes Field
68
Fill in Signature Byte Field
70
Fill in Checksum Byte Field
72
Set Beginning Disk Cluster Field to 0
74
Fill in Filename Bits
76
Return

Figure 5a

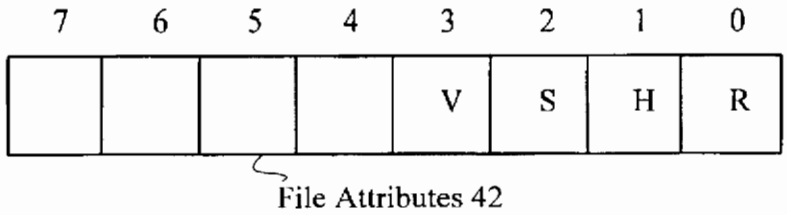


Figure 5b

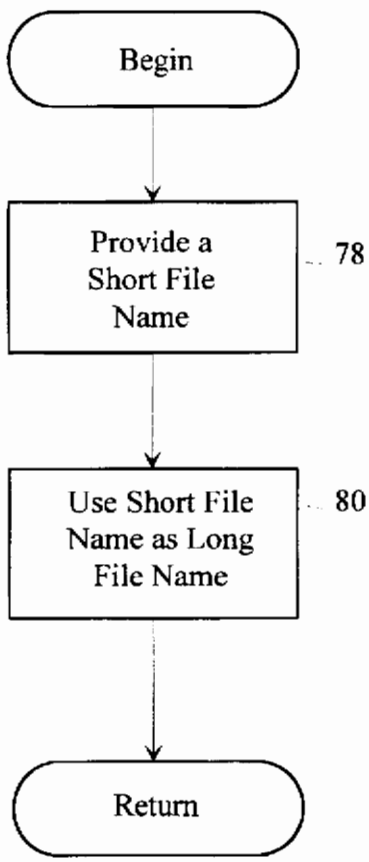


Figure 6a

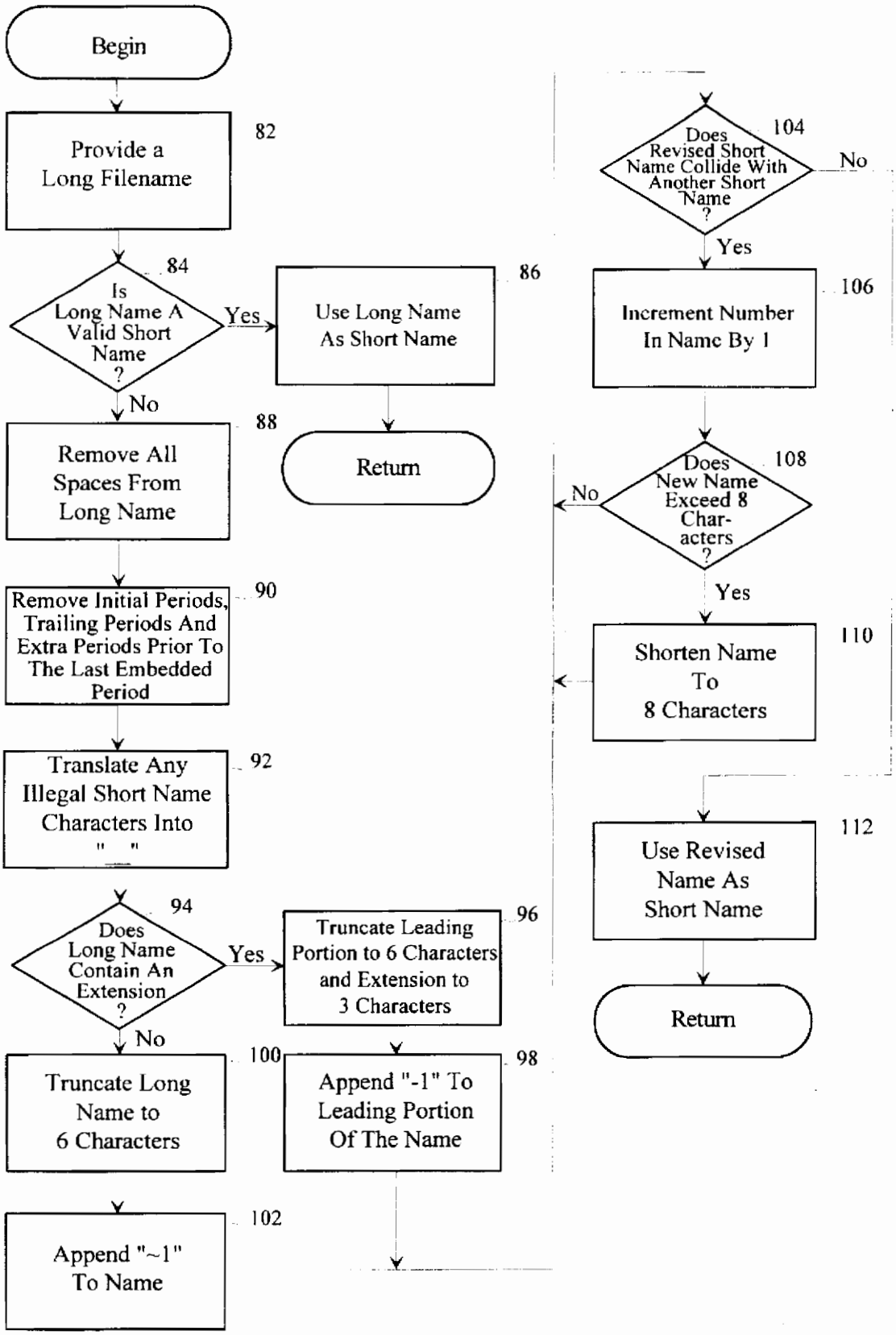

Begin
Provide a Long Filename
82
Is Long Name A Valid Short Name ?
84
Yes
Use Long Name As Short Name
86
Return
No
Remove All Spaces From Long Name
88
Remove Initial Periods, Trailing Periods And Extra Periods Prior To The Last Embedded Period
90
Translate Any Illegal Short Name Characters Into " "
92
Does Long Name Contain An Extension ?
94
Yes
Truncate Leading Portion to 6 Characters and Extension to 3 Characters
96
No
Truncate Long Name to 6 Characters
100
Append "-1" To Leading Portion Of The Name
98
Append "~1" To Name
102
Does Revised Short Name Collide With Another Short Name ?
104
No
Yes
Increment Number In Name By 1
106
Does New Name Exceed 8 Char-acters ?
108
No
Yes
Shorten Name To 8 Characters
110
Use Revised Name As Short Name
112
Return

Figure 6b

COMMON NAME SPACE FOR LONG AND SHORT FILENAMES

This application is a continuation of U.S. patent application Ser. No. 081427,004, filed Apr. 24, 1995, now U.S. Pat. Ser. No. 5,579,517 which is a file wrapper continuation of U.S. patent application Ser. No. 08/041,497, filed Apr. 1, 1993, now abandoned.

TECHNICAL FIELD

The present invention relates generally to data processing systems and, more particularly, to a common name space for long and short filenames.

BACKGROUND OF THE INVENTION

Many operating systems, such as the MS-DOS, version **5**, operating system, sold by Microsoft Corporation of Redmond, Wash., support only short filenames. In the MS-DOS, version **5**, operating system, filenames may be a maximum length of eleven characters. Each filename may have a main portion of eight characters followed by an extension of three characters. An example filename in the MS-DOS, version **5**, operating system is "EXAMPLE1.EXE", wherein "EXAMPLE1" constitutes the main portion and "EXE" constitutes the extension.

Each filename is limited to eleven characters due to constraints in the file system of the MS-DOS, version **5**, operating system. This file system employs a directory structure in which each file has a directory entry associated with it. Unfortunately, the directory entry for a file only supports filenames with a maximum length of eleven characters. Such a limit in the length of the filenames is often frustrating to a user. The length limit of eleven characters prevents a user from employing adequately descriptive filenames and, in many instances, forces a user to insert awkward abbreviations of descriptive terms into the filename.

SUMMARY OF THE INVENTION

It is, therefore, an object of the present invention to provide a system that supports long filenames.

It is another object of the present invention to provide a system that supports long filenames while minimizing the compatibility impact of supporting long filenames.

It is a further object of the present invention to provide a system that supports a common name space for both long filenames and short filenames.

In accordance with the first aspect of the present invention, a method is practiced in a data processing system having a memory means and a processing means. In accordance with this method, a first directory entry is created and stored in the memory means for a file. The first directory entry holds a first filename for the file and information about the file. A second directory entry is also created and stored in the memory means. The second directory entry holds at least one portion of a second filename having a fixed number of characters and information about the file. One of the first or second directory entries is accessed in the memory means to gain access to the information contained therein.

In accordance with another aspect of the present invention, a data processing system includes a memory that holds a first directory entry for a file, a second directory entry for the file, and an operating system. The first directory entry includes a first filename for the file and the second directory entry includes the second filename for the file. The second filename includes more characters than the short filename. The data processing system also includes a processor for running the operating system and accessing either the first directory entry or the second directory entry to locate the file.

In accordance with yet another aspect of the present invention, a method is practiced in a data processing system having memory. In accordance with this method, a file is created and the file is assigned a user-specified long filename. The long filename is manipulated with the data processing system to create a short filename of fewer characters. The long filename and the short filename are stored in memory so that the file can be accessed by either the long filename or the short filename.

In accordance with a further aspect of the present invention, a method is practiced in which a first directory entry for a file is stored in a memory means. The first directory entry holds the short filename for the file. The short filename includes at most a maximum number of characters that is permissible by an application program. A second directory entry is also stored in the memory means for the file. A second directory entry holds at least the first portion of a long filename for the file. The long filename includes a greater number of characters than the maximum number of characters that is permissible by the application program. The application program is run on a processor of the data processing system. The application program identifies the file by the short filename.

In accordance with a still further aspect of the present invention, a method is practiced in which a first directory entry is stored in the memory means for a file. The first directory entry holds a short filename for the file that includes at most the maximum number of characters that is permissible by the operating system. A second directory entry is stored in the memory means for the file. The second directory entry holds a long filename for the file that includes more than the maximum number of characters that is permissible by the operating system. In this instance, the operating system does not use long filenames; rather, it uses solely short filenames. The first directory entry is accessed by the operating system to locate the file.

BRIEF DESCRIPTION OF THE DRAWINGS

A preferred embodiment of the present invention will now be described herein with reference to the Drawings. The Drawings include the following Figures.

FIG. **1** is a block diagram of a data processing system used for implementing the preferred embodiment of the present invention.

FIG. **2** is a block diagram illustrating the storage of short filename directories in locations adjacent to long filename directory entries.

FIG. **3***a* shows the format of a short filename directory entry in the preferred embodiment of the present invention.

FIG. **3***b* shows the format of a long filename directory entry in the preferred embodiment of the present invention.

FIG. **4** is a flow chart illustrating the steps performed by the preferred embodiment of the present invention when a new file is created.

FIG. **5***a* is a flow chart illustrating the steps performed in creating a long filename directory entry in the preferred embodiment of the present invention.

FIG. **5***b* is a block diagram illustrating bits in the file attributes fields of the long filename directory entry of FIG. **3***b*.

FIG. 6*a* is a flow chart illustrating the steps performed when a short filename is provided by the user in the preferred embodiment of the present invention.

FIG. 6*b* is a flow chart illustrating the steps performed when a long filename is provided by user in the preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

A preferred embodiment of the present invention described herein provides support for the use of long filenames (i.e., filenames that may have substantially more characters than current operating systems, such as the MS-DOS, version **5**, operating system permit). "Short filenames" will be used hereinafter to refer to filenames that have a small limit (such as 11 characters) as to the maximum number of characters permitted. In the preferred embodiment, the long filenames are provided in a common name space with the short filenames. A long filename and a short filename are provided for each file in the system. The sharing of a common name space is realized through providing separate directory entries for long filenames and short filenames. Each file has a short filename directory entry associated with it and may also have at least one long filename directory entry. The short filenames are like those provided previously in the MS-DOS, version **5**, operating system. The long filenames, as will be described in more detail below, may have a maximum length of up to 255 characters. The preferred embodiment will be described with reference to an implementation with the MS-DOS, version **5**, operating system.

The potential compatibility problems of supporting long filenames are apparent by considering one solution to the problem of short filenames. This solution is not part of the present invention and is described herein merely to illustrate how a preferred embodiment avoids the compatibility problems suffered by this proposed solution. This solution supports long filenames by merely increasing the number of characters the operating system permits for a filename.

There are two major difficulties with this solution. First, the existing application bases of many systems use only short filenames (e.g., 11 characters or less) and are not prepared to utilize only long filenames (e.g., up to 255 characters). As an example, an application may allocate a buffer large enough to hold the short filename and if the operating system tries to place long filename data into this buffer, the buffer may overflow so as to cause the application data to be unexpectedly overwritten. Second, certain disk utility programs access the file system volume directly and, thus, do not rely on the operating system to access the files. If the file system is changed to support long filenames, compatibility problems with the disk utility programs arise.

The preferred embodiment of the present invention described herein, in contrast, seeks to minimize the compatibility impact of supporting long filenames by providing both a long filename and a short filename for each file. As a result, applications and utilities that require short filenames still have short filenames available, and applications that use long filenames have long filenames available.

The preferred embodiment of the present invention may be implemented as code realized as software or firmware. In order to support long filenames, the preferred embodiment of the present invention provides several long filename application program interfaces (APIs). These APIs are provided along with the conventional short filename interfaces that are standard with the operating system. The long filename APIs support file operations and directory entries for long filenames. The APIs include a file attributes function, a file delete function, a file directory function, a file find function, a file open/create function and a file rename function.

The preferred embodiment of the present invention may be implemented in a data processing system **10** like that shown in FIG. **1**. This data processing system **10** includes a central processing unit (CPU) **12** with a standard set of registers **13** that includes an accumulator (AL) register **15**, a memory **16** and input/output (I/O) devices **14**. The CPU **12** oversees the operations of the data processing system **10** and has access to the memory **16** and the I/O devices **14**. The memory **16** may include both RAM and disc storage. The memory **16** holds an operating system **17** (denoted as "**0.S.**" in FIG. **1**) which includes the long and short filename APIs. Those skilled in the art will appreciate that the present invention may be implemented on other suitable data processing systems.

All of the functions for the long filename APIs and short filename APIs are incorporated into the operating system **17**. Those functions are supported through an Int **21***h* interrupt call (where **21***h* denotes 21 in hexadecimal notation). In other words, all the functions are called by executing an Int **21***h* interrupt, wherein the function that is called through the Int **21***h* interrupt is specified by a value placed in a register, as will be described in more detail below. The Int **21***h* interface is like that provided in the MS-DOS, version **5**, operating system except that the interface also supports calls to functions for long filenames. In calls to the long filename APIs, the function number to be called is placed in the AL register **15** of a processor, such as the CPU **12** in FIG. **1** before the interrupt is initiated.

In order to support both a long filename and a short filename for each file, the preferred embodiment provides a short filename directory entry **18** (FIG. **2**) and may provide at least one long filename directory entry **20** for each file in a common name space. Each file has a long filename and a short filename associated with it. A long filename directory entry **20** is only created when the long filename cannot be correctly stored in the short filename directory entry. The long filename directory entries **20** are stored adjacent to the corresponding short filename directory entry **18** as part of the common name space used in memory **16**. Moreover, the long filename directory entries **20** are configured to minimize compatibility problems with operating systems that support only short filenames.

FIG. **2** shows an example of the directory entries **18** and **20** for a file in the preferred embodiment described herein. The short filename directory entry **18** is provided along with several long filename directory entries **20**. The number of long filename directory entries **20** provided (including zero long filename directory entries) for a file depends upon the number and type of characters in the long filename. As will be described in more detail below, each long filename directory entry **20** may hold up to 26 characters of a long filename. The long filename directory entries **20** are dynamically allocated based upon the number of characters in the long filename. For example, a file with a long filename of 50 characters has two long filename directory entries **20** allocated for it, whereas a file with a long filename of 70 characters has three long filename directory entries **20** allocated for it. As was mentioned above, a long filename may have a maximum of 255 characters and thus, a maximum of 10 long filename directory entries **20** may be allocated for any file. The maximum of 255 characters per filename is a product of maximum path length (260 characters) limitations of the operating system **17**.

There may be many instances in which the long filename does not completely fill all of the space available in the allocated long filename directory entries **20**. In such an instance, a null terminator is placed after the last character of the long filename so that additional spaces or nonsensical data will not be returned. The extra spaces are filled with OFFh (where "h" indicates the use of hexadecimal notation).

FIG. **3***a* illustrates the format of the short filename directory entry **18**. Each of the fields in the directory entry begins at a different offset relative to the starting address of the directory entry. A filename field **22** holds the main portion (i.e., the leading 8 characters) of the short filename. As the main portion of the short filename may hold up to eight characters of the short filename, the filename field **22** is eight bytes in length and begins at offset **00***h*. The filename field **22** is followed by a file extension field **24** at offset **08***h*. The file extension field holds the characters of the extension of the short filename. The extension field **24** is three bytes in length (encoding three characters).

Following the extension field **24** at offset OBh is a file attributes field **26**. The file attributes field **26** includes a number of bits that, based upon whether the bits are set or not, specify information about the associated file.

The short filename directory entry **18** also includes a reserved field **28**. The reserved field **28** begins at offset OCh and is ten bytes in length. The short filename directory entry **18** additionally includes a time of last update field **30** and a date of last update field **32**. The time of last update field **30** is two bytes in length and begins at offset **16***h*. The date of last update field **32** is two bytes in length and begins at offset **18***h*.

The short filename directory entry **18** includes a beginning disk cluster field **34**. The beginning disk cluster field **34** holds a pointer to the section of the memory **16** (FIG. **1**) where the file's first disk cluster is held (i.e. to the beginning of the allocation chain for the file). This beginning disk cluster field **34** (FIG. **3***a*) is stored at offset 1Ah and is two bytes in length. A file size field **36** follows the beginning disk cluster field **34**. The file size field **36** holds a value that specifies the amount of memory occupied by the file associated with the short filename directory entry **18**. The file size field **36** is four bytes in length and begins at offset 1Ch.

FIG. **3***b* illustrates the format used for each of the long filename directory entries **20**. The long filename directory entry **20** additionally includes a signature field **38** that holds a digital signature. The signature field **38** is useful in specifying the order of a long filename directory entry **20** in a sequence of associated long filename directory entries. For example, a first long filename directory entry includes a signature field **38** that specifies that it is the first entry, and each successive long filename directory entry includes a signature field **38** that specifies where the long filename directory entry fits in the sequence of long filename directory entries for a file. The signature field **38** is provided primarily for use with utility programs that directly access the file system volume. The signature field **38** is one byte in length and begins at offset **00***h*, which is the beginning of the filename field **22** (FIG. **3***a*) of the short filename directory entry **18**. The signature field **38**, given its location in the long filename directory entry, might easily be mistaken for a portion of a short filename by certain utility programs. Hence, the signature field **38** includes only illegal short filename characters so that the characters may not be readily changed by systems or utilities that support only short filenames.

The long filename directory entry **20** includes three fields **40**, **48** and **52** that are provided for holding characters of the long filename. The first long filename field **40** begins at offset **0***lh* and may hold up to ten characters of the long filename (i.e., it is 10 bytes in length). The second long filename field **48** begins at offset OEh and may hold up to twelve characters (i.e., 12 bytes) of the long filename. Lastly, the third long filename field **52** begins at offset 1Ch and may hold up to four characters (i.e., 4 bytes) of the long filename. Thus, cumulatively, these three fields **40**, **48** and **52** may hold up to twenty-six characters of the long filename. The long filename fields **40**, **48** and **52** are filled sequentially beginning with field **40** and then filling fields **48** and **52**, consecutively.

While the long filename directory entry **20** differs from the short filename directory entry **18**, the long filename directory entry **20**, nevertheless, includes certain similar fields at the same specified offsets as were discussed above for the short filename directory entry **18** (FIG. **3***a*). As such, operating systems that do not support long filenames are not disturbed by the long filename directory entries **20**. For instance, the long filename directory entry **20** includes a file attributes field **42** which is like the file attributes field **26** (see FIG. **3***a*) provided in the short filename directory entry.

The long filename directory entry **20** contains a checksum field **44**, which is one byte in length and at offset **0**Dh. The checksum field **44** holds a checksum of the short filename. The checksum byte, as will be described in more detail below, is used to ensure that the long name is valid for the associated short filename and to act as a pointer to the short filename directory entry **18** that is helpful to disk utility programs. A flags field **43** is held at offset OCh. The flags field **43** holds a flag bit that may be set when unicode characters are used. In addition, the beginning disk cluster field **50** (FIG. **3***b*) of the long filename directory entry **20** is analogous to the beginning disk cluster field **34** (FIG. **3***a*) of the short filename directory entry **18**. However, it always has a constant value of zero in the long filename directory entry.

The above discussion has focused on how the directory entries **18** and **20** (FIG. **2**) are used to support both long filenames and short filenames. The discussion below will focus on how such directory entries are supported by the preferred embodiment of the present invention.

When a new file is created, the preferred embodiment must take steps to support both a long filename and a short filename for the new file. In discussing how the preferred embodiment supports both long filenames and short filenames, it is helpful to first focus on the creation of the directory entries and then to focus on the creation of the filenames. FIG. **4** is a flowchart depicting the basic steps performed upon creation of the new file. Initially, the new file is created (step **54**) using either a long filename API or a short filename API. Both varieties of APIs support the creation of files. Depending on the type of API that is used to create the files, the file will initially have a long filename and/or a short filename. In other words, if a file is created with a long filename API, it will initially have a long filename and if a file is created with a short filename API, it will initially have a short filename, which may also be the long filename for the file.

At least one long filename directory entry **20** may be created for the file. First, a determination is made whether a long filename directory entry **20** is required (step **51**). If the long filename will not correctly fit in the short filename directory entry **18**, a long filename directory entry **20** is required. Long filename directory entries **20** are dynamically allocated based upon the number of characters in the long filename. At a minimum, a short filename directory entry **18**

will be created that has the format that is shown in FIG. 3*a*. Thus, the system checks to see how many long filename directory entries are needed and allocates space for the short filename directory entry and as many additional long filename directory entries as are required (step **58**). It should be appreciated that when both a short filename directory entry **18** and at least one long filename directory entry **20** are created, space for both types of directory entries are allocated together at the same time. The long and short filename directory entries **18** and **20** are then filled in step **59**. However, if no long filename directory entry is required, no space will be allocated (i.e., steps **58** and **59** are skipped).

FIG. **5***a* is a flowchart depicting the steps performed in filling in a long filename directory entry **20** (see step **59** in FIG. **4**). The steps are shown in a given sequence, but those skilled in the art will appreciate that the steps need not be performed in this illustrated sequence. Rather, other sequences are equally acceptable.

A hidden bit in the file attributes field **42** is set to have a value of one (step **62**). FIG. **5***b* shows the bits included in the file attributes field **42**. The hidden bit is designated by the letter "H" in FIG. **5***b* and is present at bit position **1** in the file attributes field **42**. When the hidden bit is set to a value of one, the directory entry is hidden and is excluded from normal searches of the directory entries **18** and **20**. By setting the hidden bit, the long filename directory entries **20** (FIG. **2**) are not searched in conventional directory entry searches. The hidden bit is set so that down level systems (i.e., systems that support only short filenames) will not see the long filename directory entries **20**.

A read-only bit is also set in the file attributes field (step **64** in FIG. **5***a*). The read-only bit is designated by the letter "R" in FIG. **5***b* and is present at bit position **0** in the file attributes field **42**. Setting the read-only bit to a value of one indicates that the file is a read-only file and any attempts to write to the file will fail.

A system bit in the file attributes field **42** is set to a value of one (step **66** in FIG. **5***a*). The system bit is designated by the letter "S" in FIG. **5***b* and is present at bit position **2** in the file attributes field **42**. Setting the system bit to a value of one designates the file as a system file and excludes the directory entry from normal searches of the directory entries **18** and **20**. The setting of the system bit to a value of one hides the long filename directory entries **20** from down level operating systems that support only short filenames.

Next, a volume label bit is set in the file attributes field **42** (step **68** in FIG. **5***a*). The volume label bit is designated by the letter "V" in FIG. **5***b* and is present at bit position **3** in the file attributes field **42**. Setting the volume label bit to a value of one hides the long filename directory entry from "Check Disk" operations of certain disk utility programs. For example, MS-DOS, version **5.0**, includes a utility named CHKDSK. The setting of the volume label attribute hides the long filename directory entries from CHKDSK.

The discussion will now return again to the flowchart of FIG. **5***a*. The signature byte field **38** (FIG. **3***b*) is filled with a digital signature (step **70** in FIG. **5***a*). As was mentioned above, the signature distinguishes the order of the long filename directory entries **20** for the file. The checksum field **44** in FIG. **3***b* is filled with the appropriate checksum of the short filename (step **72** in FIG. **5***a*). The checksum byte field **44** (FIG. **3***b*) is used to associate the long filename directory entries **20** with their appropriate short filename by holding a checksum of the short filename. The beginning disk cluster field **50** (FIG. **3***b*) is set to zero (step **74** in FIG. **5***a*). The long filename directory entry **20**, thus, has no data allocated to it. This helps to make the long filename directory entry invisible in down level systems. Lastly, the bits for the characters of the long filename are stored in the appropriate long filename fields **40**, **48** and **52** (FIG. **3***b*) of the long filename directory entry **20** (step **76** in FIG. **5***a*).

By setting the file attributes field **42** (FIG. **5***b*) bits as described above and by setting the beginning disk cluster field **50** to zero (FIG. **3***b*), the preferred embodiment of the present invention makes the long filename directory entries nearly invisible to operating systems that support only short filenames (i.e., down level systems). Nevertheless, files with long filenames are still permitted in down level operating systems. The long filename directory entries are not visible in the directory entry listing for down level systems. The combination of bit settings in the file attributes field and the zeroing of the beginning disk cluster field **50** make the long filename directory entries invisible to down level systems. Thus, compatibility problems arising from having long filenames in the down level operating system are minimized. Moreover, utility programs, that may skew the order of directory entries, are not a problem. The signature field **40** (FIG. **3***b*) and the checksum field **44** may be used in conjunction to rearrange entries that are out of order. In particular, the checksum fields **44** are used to associate long filename directory entries **20** with a short filename directory entry and the signature fields **40** of the long filename directory entries are used to assign related long filename directory entries into proper sequence.

The discussion above has noted that filenames are created using either short filename APIs or long filename APIs. As a result, when a file is created it has either a long filename or short filename assigned to it by the user, depending on whether a long filename API or short filename API is used. The preferred embodiment of the present invention described herein automatically creates the missing short filename or long filename. For instance, if a file is created using a short filename API, the preferred embodiment described herein establishes a corresponding long filename (which is the same as the short filename). Analogously, if a file is created using a long filename API, the preferred embodiment generates a corresponding short filename that is stored in a short filename directory entry **18**. FIG. **6***a* shows the steps performed by the preferred embodiment when the short filename is provided by the user. In particular, the user provides a short filename (step **78** in FIG. **6***a*), and the short filename is used as the long filename (step **80**). When the user provides a short filename, the system checks whether the name is a valid short filename and whether there are any existing files that pose a conflict (not shown). If there is no problem in terms of format or conflict, the file is assigned the provided short filename. The short filename is then used as the long filename, and there is no long filename directory entry **20** for the file.

When a file is created using a long filename API, the resulting creation of a corresponding short filename may be quite complex. FIG. **6***b* is a flowchart illustrating the steps performed to create the short filename in such an instance. Initially, the long filename is provided by the user (step **82** in FIG. **6***b*). The preferred embodiment then checks whether the long filename is a valid short filename (step **84**). If the long filename is a valid short filename, the long filename is used as the short filename (step **86**).

However, if the long filename does not qualify as a valid short filename, a short filename is created by removing the spaces from the long filename and using the resulting characters as a proposed short filename (step **88**). Initial periods, trailing periods and extra periods that are prior to

the last embedded period are then removed from the proposed short filename (step **90**). Furthermore, any illegal short filename character is translated into an underscore (step **92**). A check of whether the proposed short filename contains an extension is then performed (step **94**). If the proposed short filename contains an extension, the leading main portion of the filename is truncated to six characters in length, and the leading three characters of the extension are used (step **96**). Subsequently, a "~1" is appended to the leading portion of the remaining characters (step **98**) to serve as the short filename.

If the modified long filename does not contain an extension (step **94**), the long filename is truncated to six characters (step **100**), and "~1" is appended to the truncated filename (step **102**) to serve as the short filename. In both of the above-described instances (i.e., the "yes" instance and "no" instance of step **94**), the preferred embodiment next checks whether the proposed short filename collides with any other short filename (step **104**). If the proposed short filename does not collide with another short filename (i.e., there is no other identical short filename), the proposed short filename is assigned as the short filename for the file (step **112**). In the case where the proposed short filename collides with another short filename, the characters that are appended to the name are incremented by one (step **106**). Thus, if the number value is initially "~1", the number value is incremented in step **106** by one to "~2". The preferred embodiment checks whether the new proposed short filename exceeds eight characters in length (step **108**). If the new proposed short filename does not exceed eight characters in length, the checking of whether the proposed short filename collides with another short filename is repeated (step **104**). When the number of characters in the filename exceeds eight characters in length, the new short filename is shortened to eight characters (step **110**). In particular, if the length of the leading portion of the filename (ignoring the extension) plus the tilda and the number exceeds eight characters, the leading portion of the filename is shortened until the new proposed short filename (absent the extension) fits in eight characters. For example, the filename "MonKey~10.EXE" is shortened to "MonKe~10.EXE." The above-described steps **104**, **106**, **108** and **110** are repeated until a short filename is created for the file that is of proper length and that does not collide with another short filename.

The preferred embodiment of the present invention provides a solution to the problem of short filenames while minimizing the compatibility impact of the solution. The use of a common name space that provides a long filename and a short filename for each file allows the files to be used both with applications that support short filenames and applications that support long filenames.

While the present invention has been described with reference to a preferred embodiment thereof, those skilled in the art will appreciate that various changes in scope and form may be made without departing from the present invention as defined in the appended claims.

We claim:

1. In a computer system having a storage, a directory service for accessing directory entries and a file system that uses the directory entries to access files, a method, comprising the computer-implemented steps of:

(a) creating a first directory entry for a file wherein the first directory holds a short filename for the file and the location of the file;

(b) creating a second directory entry for the file wherein the second directory entry holds at least one portion of a long filename having a fixed number of characters and a signature that identifies that the second directory entry holds a first portion of the long filename;

(c) storing the first directory entry and the second directory entry on the storage among the directory entries used by the directory service; (d) accessing the second directory entry by the directory service to access the file; and (e) creating and storing in the storage a sequence of at least one additional directory entry for holding a next sequential portion of the long filename.

2. The method as recited in claim **1** wherein the long filename contains more characters than the short filename.

3. The method as recited in claim **1** wherein each additional directory entry may hold only a fixed number of characters of the long filename and how many additional directory entries are created is dictated by how many additional directory entries are required to store characters of the long filename which are not already stored in the second directory entry.

4. The method as recited in claim **1** wherein the step of creating at least one additional directory entry for the long filename further comprises the step of creating a plurality of additional directory entries.

5. The method as recited in claim **1** wherein the step of creating at least one additional directory entry for the long filename further comprises the step of providing a signature in each additional directory entry that identifies which portion of the long filename the additional directory entry holds.

6. The method as recited in claim **1** wherein the step of creating at least one additional directory entry for the long filename further comprises the step of providing a checksum of the first filename in each additional directory entry.

7. In a data processing system having a processor running an operating system and a memory means having memory locations wherein the operating system is stored in the memory means, a method, comprising the steps of:

(a) storing in a first of the memory locations of the memory means a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters that is permissible by an application program;

(b) storing in a second of the memory locations of the memory means that is adjacent to the first of the memory locations a second directory entry for the file wherein the second directory entry holds at least a first portion of a long filename for the file, said long filename including a greater number of characters than the maximum number of characters that is permissible by the application program, and

(c) accessing one of the directory entries to locate the file.

8. The method as recited in claim **7** wherein the step of storing the second directory further comprises the step of storing a checksum of the short filename in the second directory entry.

9. The method as recited in claim **7**, further comprising the step of storing at least one additional directory entry holding a next portion of the long filename in the memory means.

10. The method as recited in claim **9** wherein the step of storing at least one additional directory entry further comprises the step of storing a checksum of the short filename in the additional directory entry.

11. The method as recited in claim **9** wherein the step of storing at least one additional directory entry further comprises the step of storing a signature that uniquely identifies

which portion of the long filename is stored in the additional directory entry.

12. In a computer system having a storage, a directory service for accessing directory entries and a file system that uses the directory entries to access files, a comnputer-readable medium holding computer-executable instructions for performing a method comprising computer-implementented steps of:

(a) creating a first directry entry for a file wherein the first directory holds a short filename for the file and the location of the file;

(b) creating a second directory entry for the file wherein the second directory entry holds at least one portion of a long filename having a fixed number of characters;

(c) storing the first directory entry and the second directory entry on the storag among the directory entries used by the directory service; and

(d) accessing the second directory entry by the directory service to access the file.

13. The computer-readable medium of claim **12** wherein the long filename contains more characters than the short filename.

14. The computer-readable medium of claim **12** also holding computer-executable instructions for creating and storing in the storage a sequence of at least one additional directory entry for holding a next sequential portion of the long filename.

15. The computer-readable medium of claim **14** wherein each additional directory entry may hold only a fixed number of characters of the long filename and how many additional directory entries are created is dictated by how many additional directory entries are required to store characters of the long filename which are not already stored in the second directory entry.

16. The computer-readable medium of claim **14** wherein the step of creating at least one additional directory entry for the long filename further comprises the step of creating a plurality of additional directory entries.

17. The computer-readable medium of clain **14** wherein the step of creating the second directory entry further comprises the step of providing a signature in the second directory entry that identifies that the second directory entry holds the first portion of the long file name.

18. The computer-readable medium of claim **17** wherein the step of creating at least one additional directory entry for the long filename further comprises the step of providing a signature in each additional directory entry that identifies which portion of the long filename the additional directory entry holds.

19. The computer-readable medium of claim **14** wherein the step of creating at least one additional directory entry for the long filename further comprises the step of providing a checksum of the first filename in each additional directory entry.

20. In a data processing system having a processor running an operating system and a memory means with memory locations, wherein said memory means stores the operating system, a computer-readable medium holding computer-executable instructions for performing a method comprising the steps of:

(a) storing in a first of the memory locations of the memory means a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters that is permissible by an application program;

(b) storing in a second of the memory locations of the memory means that is adjacent to the first of the memory locations a second directory entry for the file wherein the second directory entry holds at least a first portion of a long filename for the file, said long filename including a greater number of characters than the maximum number of characters that is permissible by the application program; and

(c) accessing one of the directory entries to locate the file.

21. The computer-readable medium of claim **20** wherein a checksum of the short filename is stored in the second directory entry.

22. The computer-readable medium of claim **20** wherein at least one additional directory entry is stored to hold a next portion of the long filename in the memory means.

23. The computer-readable medium of clain **22** wherein a signature is stored in the additional directory entry that uniquely identifies which portion of the long filename is stored in the additional directory entry.

24. In a computer system having a directory service for accessing directory entries and a file system that uses the directory entries to access files, a method comprising the computer-implemented steps of:

(a) creating a first directory entry for a file wherein the first directory entry holds a short filename for the file and the location of the file.

(b) creating a second directory entry for a file wherein the second directory entry is configured to appear as if it holds a short filename to a program that uses only short filenames and wherein the second directory entry holds at least one portion of a long filename for the file, said long filename having more characters then the short filename; and

(c) accessing one of the first directory entries and the second directory entry by the directory service in order to access the file.

25. The method of claim **24** wherein the program that uses only short filenames is an operating system.

26. The method of claim **24** wherein the program that uses only short filenames is an application program.

27. The method of clain **24** wherein the storage includes storage locations and wherein the first directory entry and the second directory entry are stored in adjacent storage locations.

28. In a computer system having a directory device for accessing directory entries and a file system that uses the directory entries to access files, a computer-readable medium holding computer-executable instructions for executing a method comprising the computer-implemented steps of:

(a) creating a first directory entry for a file wherein the first directory entry holds a short filename for the file and the location of the file;

(b) creating a second directory entry for a file wherein the second directory entry is configured to appear as if it holds a short filename to a program that uses only short filenames and wherein the second directory entry holds at least one portion of a long filename for the file, said long filename having more characters then the short filename; and

(c) accessing one of the first directory entries and the second directory entry by the directory service in order to access the file.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,758,352

DATED : May 26, 1998

INVENTOR(S) : A.R. Reynolds et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

TITLE PAGE, ITEM	LINE	
[56]	Refs. Cited (Other Pubs., Item 4)	"sharing" should read --Sharing--
[56]	Refs. Cited (Other Pubs., Item 5)	"1988" should read --1989--
[56]	Refs. Cited (Other Pubs., Item 8)	after "Duncan" insert --,--
COLUMN		
10 (Claim 1,	6 line 15)	after "service;" insert paragraph return
10 (Claim 1,	8 line 17)	after "file; and" insert paragraph return
10 (Claim 7,	51 line 18)	"program," should read --program;--
11 (Claim 12,	8 line 6)	"implementented" should read --implemented--
11 (Claim 12,	9 line 7)	"directry" should read --directory--
11 (Claim 12,	16 line 14)	"storag" should read --storage--
11 (Claim 17,	43 line 5)	"file name" should read --filename--

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,758,352
DATED : May 26, 1998
INVENTOR(S) : A.R. Reynolds et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

COLUMN	LINE	
12 (Claim 23,	16 line 1)	"clain" should read --claim--
12 (Claim 24,	27 line 7)	"," should read --;--
12 (Claim 24,	33 line 13)	"then" should read --than--
12 (Claim 27,	42 line 1)	"clain" should read --claim--
12 (Claim 28,	60 line 15)	"then" should read --than--

Signed and Sealed this

Twenty-ninth Day of September, 1998

Attest:

BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks

US005758352C1

(12) EX PARTE REEXAMINATION CERTIFICATE (5551st)

United States Patent

Reynolds et al.

(10) Number: US 5,758,352 C1

(45) Certificate Issued: *Oct. 10, 2006

(54) **COMMON NAME SPACE FOR LONG AND SHORT FILENAMES**

(75) Inventors: **Aaron R. Reynolds**, Redmond, WA (US); **Dennis R. Adler**, Mercer Island, WA (US); **Ralph A. Lipe**, Woodinville, WA (US); **Ray D. Pedrizetti**, Issaquah, WA (US); **Jeffrey T. Parsons**, Redmond, WA (US); **Rasipuram V. Arun**, Redmond, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

Reexamination Request:
No. 90/007,372, Jan. 8, 2005

Reexamination Certificate for:

Patent No.:	**5,758,352**
Issued:	**May 26, 1998**
Appl. No.:	**08/711,692**
Filed:	**Sep. 5, 1996**

(*) Notice: This patent is subject to a terminal disclaimer.

Certificate of Correction issued Sep. 29, 1998.

Related U.S. Application Data

(63) Continuation of application No. 08/427,004, filed on Apr. 24, 1995, now Pat. No. 5,579,517, which is a continuation of application No. 08/041,497, filed on Apr. 1, 1993, now abandoned.

(51) **Int. Cl.**
G06F 12/00 (2006.01)
G06F 17/30 (2006.01)

(52) **U.S. Cl.** **707/200**; 707/1; 707/6

(58) **Field of Classification Search** 707/1, 707/6, 200

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,058,672	A	* 11/1977	Crager et al.	370/394
4,780,821	A	10/1988	Crossley	718/100
4,945,475	A	7/1990	Bruffey et al.	707/1
4,945,476	A	7/1990	Bodick et al.	600/301
4,987,531	A	1/1991	Nishikado et al.	707/200
4,999,766	A	3/1991	Peters et al.	707/10
5,058,000	A	10/1991	Cox et al.	707/10

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0 462 587 B1	12/1996
EP	0 578 205 B1	3/2000
EP	0 618 540 B1	12/2001
JP	64-41039	2/1989
JP	1 315 843	12/1989
JP	2 148 341	7/1990
JP	4-297 934	10/1992
JP	6 019 763	1/1994

OTHER PUBLICATIONS

Manes, S. "Taking a Gamble with WordVision", PC Magazine, vol. 3, No. 6, pp. 211–221, Apr. 3, 1984.*

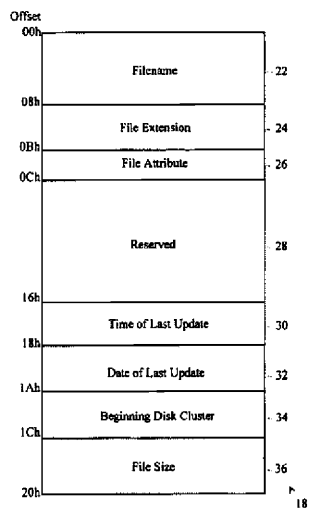
Jeffries, R. "What's Ahead for DOS", PC Magazine, vol. 4, No. 24, pp. 95–97, Nov. 26, 1985.*

(Continued)

Primary Examiner—Luke S Wassum

(57) **ABSTRACT**

An operating system provides a common name space for both long filenames and short filenames. In this common namespace, a long filename and a short filename are provided for each file. Each file has a short filename directory entry and may have at least one long filename directory entry associated with it. The number of long filename directory entries that are associated with a file depends on the number of characters in the long filename of the file. The long filename directory entries are configured to minimize compatibility problems with existing installed program bases.



U.S. PATENT DOCUMENTS

5,083,264	A		1/1992	Platteter et al.	714/5
5,129,088	A		7/1992	Auslander et al.	711/1
5,179,703	A		1/1993	Evans	717/122
5,202,982	A		4/1993	Gramlich et al.	707/2
5,202,983	A		4/1993	Orita et al.	707/4
5,287,502	A		2/1994	Kaneko	707/4
5,291,595	A		3/1994	Martins	707/200
5,307,494	A	*	4/1994	Yasumatsu et al.	707/200
5,313,646	A		5/1994	Hendricks et al.	707/101
5,317,733	A		5/1994	Murdock	707/203
5,329,427	A		7/1994	Hogdahl	361/730
5,355,497	A	*	10/1994	Cohen-Levy	707/200
5,359,724	A		10/1994	Earle	707/205
5,359,725	A		10/1994	Garcia et al.	707/200
5,363,487	A		11/1994	Willman et al.	710/8
5,367,671	A		11/1994	Feigenbaum et al.	707/1
5,371,885	A		12/1994	Letwin	707/205
5,388,257	A		2/1995	Bauer	707/1
5,412,808	A		5/1995	Bauer	707/1
5,421,001	A		5/1995	Methe	707/1
5,434,974	A		7/1995	Loucks et al.	707/101
5,437,029	A		7/1995	Sinha	707/200
5,483,652	A		1/1996	Sudama et al.	707/10
5,485,606	A		1/1996	Midgdey et al.	707/10
5,535,375	A		7/1996	Eshel et al.	703/27
5,537,592	A		7/1996	King et al.	707/200
5,596,755	A		1/1997	Pletcher et al.	710/261
5,627,996	A		5/1997	Bauer	703/20
5,694,606	A		12/1997	Pletcher et al.	710/261
5,745,752	A		4/1998	Hurvig et al.	707/200
5,745,902	A		4/1998	Miller et al.	707/200
5,754,848	A		5/1998	Hanes	707/200
5,758,352	A		5/1998	Reynolds et al.	707/200
5,761,675	A	*	6/1998	Isenberg	707/200
5,765,169	A		6/1998	Conner	707/200
5,819,275	A		10/1998	Badger et al.	707/100
5,926,805	A		7/1999	Hurvig et al.	707/2
6,055,527	A		4/2000	Badger et al.	707/2

OTHER PUBLICATIONS

Prosise, J. "Retrofitting a DOS System", PC Magazine, vol. 5, No. 21, pp. 303–313, Dec. 9, 1986.*

Sanders, S.L., ed. "The $R/O Read Only", Jun. 1987.*

Wendin Inc. "Wendin to Release Wendin–DOS in September", PR Newswire Press Release, Aug. 28, 1987.*

Mefford, M.J. "Adding Notes to Directories", PC Magazine, vol. 6, No. 15, pp. 385–394, Sep. 15, 1987.*

Petzold, C. "OS–2: A New Beginning for PC Applications", PC Magazine, vol. 7, No. 7, pp. 273–281, Apr. 12, 1988.*

Digital Research Inc. "Digital Research Introduces POMable, Single–User DOS–Compatible Operating System", Press Release, Jun. 8, 1988.*

Derfler, F.J. Jr. "Building Workgroup Solutions: AppleTalk", PC Magazine, vol. 7, No. 22, pp. 151–160, Dec. 27, 1988.*

World Software Corporation "Extend–A–Name: The 60 Character File Name Utility", User's Guide and Reference Manual, 1988.*

Duncan, R. "Comparing DOS and OS–2 File Systems", PC Magazine, vol. 8, No. 3, pp. 321–327, Feb. 14, 1989.*

Duncan, R. "Design Goals and Implementation of the New High Performance File System", Microsoft Systems Journal, pp. 1–13, Sep. 1989.*

Bonner, P. "What's In a Name?", PC Computing, vol. 2, No. 9, pp. 169–170, Sep. 1989.*

Gralla, P. "Factors Impeding the OS/2 Operating System", PC Computing, vol. 2, No. 12, p. 13, Dec. 1989.*

Greenberg, R.M. "Compress and Expand the Files on Your Hard Disk Automatically", PC Magazine, vol. 8, No. 21, pp. 299–310, Dec. 12, 1989.*

Petzold, C. "1989: The Year in Operating Systems", PC Magazine, vol. 9, No. 1, p. 172, Jan. 16, 1990.*

Duncan, R. "Getting Acquainted with the Latest Version of OS/2: 1.2 (Part 1)", PC Magazine, vol. 9, No. 6, pp. 343–346, Mar. 27, 1990.*

Duncan, R. "Getting Acquainted with the Latest Version of OS/2: 1.2 (Part 2)", PC Magazine, vol. 9, No. 7, pp. 317–321, Apr. 10, 1990.*

Duncan, R. "Using Long Filenames and Extended Attributes (Part 1)", PC Magazine, vol. 9, No. 8, pp. 317–323, Apr. 24, 1990.*

McCormick, J. "Presentation Manager Under OS/2 Encourages Lengthy Name–Calling", Government Computer News, vol. 9, No. 10, pp. 16–17, May 14, 1990.*

Duncan, R. "Using Long Filenames and Extended Attributes (Part 2)", PC Magazine, vol. 9, No. 9, pp. 305–309, May 15, 1990.*

Winship, S. "DOS Shells", PC Week, vol. 7, No. 25, p. 128, Jun. 25, 1990.*

Neuhaus, T. "Databases", PC Magazine, vol. 9, No. 12, pp. 435–437, Jun. 26, 1990.*

Jackson, P. "Apple Talk: The Need for Coexistence Between Apple Macintosh and the IBM PC", PC User, No. 145, p. 171, Nov. 7, 1990.*

Bonner, P. "GeoWorks Ensemble is a One–Size–Fits–All GUI", PC Computing, vol. 3, No. 12, pp. 45–46, Dec. 1990.*

Acerson, K.L. "WordPerfect® 5.1: The Complete Reference", Berkeley:Osborne McGraw–Hill, pp. 272–278, 593–595, 612–615, 1246. ISBN 0–07–881634–3. Z52.5.W65A26 1990.*

"CD–ROM: Rock Ridge Group Submits Preliminary CD–ROM Specs to NIST", EDGE: Work–Group Computing Report, vol. 2, No. 44, Mar. 25, 1991.*

Rock Ridge Technical Working Group Rock Ridge Interchange Protocol Version 1, Rev. 1.09, Jul. 24, 1991.*

Hayes, F. "Making CD–ROM Usable for UNIX", Unix World, vol. VIII, No. 7, p. 123, Jul. 1991.*

Somerson, P. "DOS 5.0: Microsoft Corporation's Improved Operating System", PC Computing, vol. 4, No. 7, pp. 97–113, Jul. 1991.*

Rizzo, J. "Disks of a Different Color: Running MSDOS Disks on an Apple Macintosh", MacUser, vol. 7, No. 8, pp. 231–234, Aug. 1991.*

Simon, Barry "What Do You Do To Overcome Disk Disasters", PC Magazine, vol. 10, No. 15, pp. 409–414, Sep. 10, 1991.*

Chin, C. "Using Unused Bytes in Directory Entry?", excerpt from comp.os.msdos.programmer newsgroup thread, Oct. 14, 1991.*

Proteo Technology "Proteo Technology Announces Way You Work New PC Productivity Software That Lets People Master Their PCs Without Knowing DOS", Press Release, Oct. 21, 1991.*

Young, A. "The CD–ROM Standards Frontier: Rock Ridge", CD–ROM Professional, vol. 4, No. 6, pp. 53–56, Nov. 1991.*

Zelnick, N. "Way You Work Does It Your Way", PC Magazine, vol. 10, No. 22, pp. 62–63, Dec. 31, 1991.*

Bonner, P. "Build a Document Manager Under Windows", PC Computing, vol. 4, No. 12, pp. 275–281, Dec. 1991.*

Hotch, R. "Will This Be The Way You Work?", Nation's Business, Mar. 1992.*
Prosise, J. "Tutor", PC Magazine, vol. 11, No. 5, pp. 397–399, Mar. 17, 1992.*
Bonner, P. "Windows 3.1: Faster and Smarter", PC Computing, vol. 5, No. 5, pp. 128–139, May 1992.*
Smith, G. "OS/2 2.0 Does the Job", PC Computing, vol. 5, No. 5, pp. 48–55, May 1992.*
Busch, D.D. "4DOS: DOS As You Like It", Computer Shopper, vol. 12, No. 7, pp. 698–699, Jul. 1992.*
Gralla, P. "Shareware", Computer Shopper, vol. 12, No. 9, pp. 701–703, Sep. 1992.*
Somerson, P. "Spy–Proof Your PC: 13 Igenious Ways to Keep Your System Secure", PC Computing, vol. 5, No. 9, pp. 218–237, Sep. 1992.*
Ruley, J.D. "Feature–Rich Beta at a Bargain Price", Windows Magazine, Oct. 1, 1992.*
Giovetti, A.C. "Way You Work: Personal Office", Compute!, Issue 146, p. 126, Nov. 1992.*
Rohan, R. "Golden Retriever Fetches Files in Windows", Computer Shopper, vol. 12, No. 11, p. 947, Nov. 1992.*
Microsoft Chicago Long Filename Specification, Revision 0.5, Dec. 4, 1992.*
Goh, S. "MS DOS 6.0", excerpt of a comp.os.msdos.misc newsgroup thread, Feb. 20, 1993.*
"Access Review Too Harsh", Letters Column, Windows Magazine, No. 403, p. 18, Mar. 1, 1993.*
Maird, M.K. "What are the Benefits of 4DOS?", excerpt of a comp.os.msdos.4dos newsgroup thread, Mar. 7, 1993.*
Above Software, "Above Software Introduces Golden Retriever 2.0b", Press Release, Mar. 29, 1993.*
O'Malley, C. "Fetching Desktop Files: Standalone Document Managers", Windows Sources, vol. 1, No. 2, p. 443, Mar. 1993.*
Fowler, D. "Cross Talking: Sharing Files Over a Mixed–Platform Network", Computer Shopper, vol. 13, No. 3, pp. 783–785, Mar. 1993.*
Berst, J. "Come Closer and I'll Tell You Some Secrets About Windows 4.0", Windows Magazine, No. 404, p. 43, Apr. 1, 1993.*
Mallory, J. "Breakthrough on DOS Filename Limits", Newsbytes, Apr. 12, 1993.*
Almax Software "Longer Filenames for DOS", Press Release, May 1993.*
Rettig, H. "Custom Windows Made Easy", Windows Magazine, No. 406, p. 264, Jun. 1, 1993.*
Capen, T. "The Ultimate File Manager", Corporate Computing, vol. 2, No. 6, p. 54, Jun. 1993.*
Lincoln, S. "Death to 8+3 Filenames!", excerpt from a comp.os.ms–windows.apps newsgroup thread, Jul. 9, 1993.*
Lewallen, D., F. Scot and E. Bott "The NT Desktop—Like Windows, Only Better", PC Computing, vol. 6, No. 7, pp. 124–127, Jul. 1993.*
Nilsson, B.A. "Sherlock Solves the Case of the Cryptic Windows Filenames", Computer Shopper, vol. 13, No. 7, p. 434, Jul. 1993.*
2010 Software "Sherlock 2.0 Released by 2010 Software", Press Release, Aug. 9, 1993.*
Clark, I.M. "Proposed DOS Header Record", excerpt from comp.os.msdos.programmer newsgroup thread, Aug. 11, 1993.*
Allen, J. et al. "The Vices and Virtues of MS–DOS", PC Computing, vol. 6, No. 8, pp. 27–31, Aug. 1993.*
Wagner, M. "Developers to Get Tour of 'Chicago'—New Microsoft OS", Open Systems Today, No. 136, p. 3, Nov. 8, 1993.*
Idol, C. "Sherlock", Compute, vol. 15, No. 11, p. 150, Nov. 1993.*
DeVoney, C. and R.C. Kennedy "NT Has Arrived", Windows Sources, vol. 1, No. 10, pp. 283–294, Nov. 1993.*
Davis, F.E. "NT, No Thanks; Wait for 4.0", Windows Sources, vol. 1, No. 10, pp. 105–106, Nov. 1993.*
Mathisen, T. "Novell's DOS 7 Offers Multitasking, Memory Management, and Peer–To–Peer Networking, Is it a Better DOS Than Microsoft's?", Byte Magazine, Jun. 1994.*
Olsen, M. "Student Writes Free Version of DOS", University of Wisconsin at River Falls Student Voice, Dec. 1, 1994.*
Saiedian, H. and M. Siddiqi "A Framework for the Assessment of Operating Systems for Small Computers", ACM SIGICE Bulletin, vol. 21, No. 4, pp. 2–27, Apr. 1996.*
Cluts, N.W. "Making Room for Long Filenames", Microsoft Developer Network Technology Team, downloaded from msdn.microsoft.com/library/en–us/dnfiles/html/msdn_longfile.asp?frame=true, Aug. 1996.*
Styer, E. "Disks", ACM SIGICE Bulletin, vol. 23, No. 1, pp. 22–32, Jul. 1997.*
vinDaci "Long Filename Specification", downloaded from home.teleport.com/~brainy/lfn.htm, Jan. 6, 1998.*
Mattias "25 Years of DR DOS History", downloaded from freedos.sourceforge.net/freedos/news/bits/drdos–hist.txt, Sep. 18, 2000.*
The PC Guide "File Allocation Tables", downloaded from www.pcguide.com/ref/hdd/file, Apr. 17, 2001.*
Tanenbaum, A.S. "Modern Operating Systems", Upper Saddle River:Prentice Hall, pp. 430–447, 2001.*
Hall, J. "An Overview of FreeDOS", downloaded from freedos.sourceforge.net/freedos/news/bits/article_fsm.txt, 2002.*
Beta Systems "DOS History", downloaded from www.powernet.co.za/info/DOS/His.Htm, Apr. 30 2003.*
Fuchs, C. "DOS Frequently Asked Questions", downloaded from www.drdos.new/faq, Jan. 1, 2004.*
jh "20 Years of DOS History", downloaded from freedos.sourceforge,net/freedos/news/bits/doshist.txt, Mar. 25, 2005.*
Wikipedia "Comparison of File Systems", downloaded from www.wikipedia.org, Aug. 29, 2005.*
Wikipedia "DOS", downloaded from www.wikipedia.org, Sep. 15, 2005.*
Bonner, Paul, "What's in a Name?," Sep. 1988, *PC–Computing*, vol. 2,(9), p. 169(2).
Bonner, Paul, "Build a Document Manager Under Windows," Dec. 1991, *PC–Computing*, vol. 4(12), p. 275–277, 280–283.
Comer, D. et al., "The Tilde File Naming Scheme" *IEEE 6th International Conference on Distributed Computing Systems*, Cambridge, Massachusetts, May 23, 1986, pp. 509–514.
William M. Crow, "Encapsulation of Applications in the New Wave Environment," *Hewlett–Packard Journal*, Aug. 1989, 40(4), p. 57–64.
Ray Duncan, "Design Goals and Implementation of the New High Performance File System" *Microsoft Systems Journal*, Sep. 1989, 4(5), pp. 1–13.
Ray Duncan, "Using Long Filenames and Extended Attributes" *Parts 1 & 2, PC Magazine*, Apr. 24 & May 15, 1990, 9(8,9), pp. 317–323 & 305–309.

Les Freed, "High–End PC–to–MAC LAN Solutions", *PC Magazine*, May 1992, 11(9), p. 203(8).
Glass, Brett, "Create Your Own Environment," *PC–Computing*, Oct. 1990, 3(10), 106–111.
Hurwicz, Mike, "MS–DOS 3.1 Makes It Easy to Use IBM PCs on a Network," *Data Communications*, Nov. 1985, 223–237.
Mallory, Jim, "Breakthrough on DOS Filename Limits," *Newsbytes News Network*, Apr. 12, 1993, 3 pages.
McCormick, John, "Presentation Manager Under OS/2 Encourages Lengthy Name–Calling," *Government Computer News*, May 14, 1990, 9(10), p. 16(2).
O'Malley, Chris, "Fetching Desktop Files: Standalone Document Managers," *Window Sources*, Mar. 1993, 1(2), p. 443(1).
Rohan, Rebecca, "Golden Retriever Fetches Files in Windows," *Computer Shopper*, Nov. 1992, 12(11), p. 947(1).
Samuel J. Leffler et al., "The Design and Implementation of the 4.3 BSD UNIX Operating System," *Addison–Wesley Publishing Company* 1989, Chapter 2, pp. 34–36.
Trivette, Donald B., "Utility Provides 60–Character Filenames," *PC Magazine*, Sep. 1988, 7(16), p. 56(1).
Wang, Y.E.G., "Universal File Names for Ada," *Ada Letters*, Jan./Feb. 1990, Integrated Software, Inc., New York, NY, vol. X(1), pp. 111–117.
"The Intelligent Way to Search," Oct. 1987, News Release, Dateline: Burlington, MA.
"File sharing Protocol," Microsoft Corporation, Nov. 7, 1988, 71 pages.
"World Software Corporation (WSC) Launches Extend–A–Name in Europe," *Computer Product Update*, Jul. 27, 1990.
"Above Software Introduces Golden Retriever 2.0b'," News Release, Dateline: Irvine, CA, Mar. 29, 1993.
Len, A.F. et al.,"New Improved Windows," *PC World*, Dec. 1993, 11(12), p. 252(3).
Mark G. Sobell , "A Practical Guide to the Unix System," Dec. 1989, System V Release 3 and BSD 4.3, pp. 12–14, 32, 66–69, 82–83 and 88–89.
"Appendix C How Filenames Are Converted," *Microsoft LAN Manager Services for Macintosh Administrator's Guide*, Jun. 1991, Version 1.0, for Microsoft Operating System/2, Microsoft Corporation, pp. 119–123.
"Long Filename Specification", Hardware White Paper, Designing Hardware for Microsoft® Operating Systems, Microsoft Corporation, Dec. 4, 1992, version 0.5, http://www.psdever.net/documents/lonfilename.pdf?theid=39, 19 pages.
Protest to Patent Application No. Canadian Patent Application No. 2,120,461, In The Canadian Patent Office, Submission under Section 34.1, filing of Prior Art, Sep. 24, 2004, 13 pages.
In the United States Patent and Trademark Office, In re Reexamination of 5,579,517, Declaration of Annie Pearson, Dec. 1, 2004, 2 pages.
Translation into English of Plaintiff's Grounds for Nullity, *Dr. Friedrich–Karl Boese* v. *Microsoft Corporation*, Dec. 21, 2004, 1–25.
Merkmalsanalyse Anspruch 1, Exhibit 5 of Reference 76, 1 page.
Merkmalsanalyse Anspruch 12, Exhibit 6 of Reference 76, 1 page.
Article from Newsgroup comp.archives, Exhibit 7a of Reference 27, Aug. 21, 1992, 1 page.
Article from Newsgroup comp.std.misc, Exhibit 7b of Reference 27, Mar. 21, 1991, 3 pages.
Article from Newsgroup comp.new.prod, Exhibit 7c of Reference 27, Aug. 20, 1991, 1 page.
Article from Newsgroup comp.unix.bsd and comp.os.linux, Exhibit 7d of Reference 27, Dec. 12, 1992, 1 page.
Response to Official Communication re: EP Application No. 94 105 169.0–2201 dated May 10, 2000, Exhibit 14 of Reference 76, 9 pages.
"Common Name Space for Long and Short Filenames," *Microsoft Corporation*, Exhibit 15 of Reference 76, 34 pages.
Communication Under Rule 51(4) EPC re: EP Application No. 94 105 169.0–2201, Dec. 19, 2000, Exhibit 16 of Reference 76, 39 pages.
Rock Ridge Interchange Protocol, Version 1, AN ISO 9660:1988 Compliant Approach to Providing Adequate CD–ROM Support for POSIX File System Semantics; Rock Ridge Technical Working Group, Revision 1.09, Jul. 24, 1991.
"Rock Ridge Group Submits Preliminary CD–ROM Specs To NIST—Sixteen Companies Offer Their Support" *The Florida SunFlash*, vol. 27 #11, Mar. 1991.

* cited by examiner

EX PARTE REEXAMINATION CERTIFICATE ISSUED UNDER 35 U.S.C. 307

THE PATENT IS HEREBY AMENDED AS INDICATED BELOW.

Matter enclosed in heavy brackets [ ] appeared in the patent, but has been deleted and is no longer a part of the patent; matter printed in italics indicates additions made to the patent.

AS A RESULT OF REEXAMINATION, IT HAS BEEN DETERMINED THAT:

The patentability of claims **24–28** is confirmed.

Claims **9**, **14** and **22** are cancelled.

Claims **1**, **7**, **8**, **10–12**, **15–17**, **19–21** and **23** are determined to be patentable as amended.

Claims **2–6**, **13** and **18**, dependent on an amended claim, are determined to be patentable.

New claims **29–44** are added and determined to be patentable.

1. In a computer system having a storage, a directory service for accessing directory entries and a file system that uses the directory entries to access files, a method, comprising the computer-implemented steps of:

(a) creating a first directory entry for a file wherein the first directory *entry* holds a short filename for the file and the location of the file;

(b) creating a second directory entry for the file wherein the second directory entry holds at least one portion of a long filename having a fixed number of characters and a signature that identifies that the second directory entry holds a first portion of the long filename;

(c) storing the first directory entry and the second directory entry on the storage among the directory entries used by the directory service;

(d) accessing the second directory entry by the directory service to access the file; and

(e) creating and storing in the storage a sequence of at least one additional directory entry for holding a next sequential portion of the long filename.

7. In a data processing system having a processor running an operating system and a memory means having memory locations wherein the operating system is stored in the memory means, a method, comprising the steps of:

(a) storing in a first of the memory locations of the memory means a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters that is permissible by an application program;

(b) storing in a second of the memory locations of the memory means that is adjacent to the first of the memory locations a second directory entry for the file wherein the second directory entry holds at least a first portion of a long filename for the file, said long filename including a greater number of characters than the maximum number of characters that is permissible by the application program;

(c) *storing at least one additional directory entry holding a next portion of the long filename in the memory means;* and

[(c)**]** (*d*) accessing one of the directory entries to locate the file.

8. The method as recited in claim **7** wherein the step of storing the second directory *entry* further comprises the step of storing a checksum of the short filename in the second directory entry.

10. The method as recited in claim **[9]** *7* wherein the step of storing at least one additional directory entry further comprises the step of storing a checksum of the short filename in the additional directory entry.

11. The method as recited in claim **[9** wherein the step of storing at least one additional directory entry further comprises the step of**]** *7 further comprising* storing a signature *in each of the second directory entry and the at least one additional directory entry* that uniquely identifies which portion of the long filename is stored in **[**the additional**]** *that* directory entry.

12. In a computer system having a storage, a directory service for accessing directory entries and a file system that uses the directory entries to access files, a computer-readable medium holding computer-executable instructions for performing a method comprising computer-implemented steps of:

(a) creating a first directory entry for a file wherein the first directory holds a short filename for the file and the location of the file;

(b) creating a second directory entry for the file wherein the second directory entry holds at least one portion of a long filename having a fixed number of characters;

(*c*) *creating a sequence of at least one additional directory entry for holding a next sequential portion of the long filename;*

[(c)**]** (*d*) storing the first directory entry **[**and**]** the second directory entry *and the at least one addtional directory entry* on the storage among the directory entries used by the directory service; and

(*e*) accessing the second directory entry *and the at least one additional directory entry* by the directory service to access the file.

15. The computer-readable medium of claim **[14]** *12* wherein each additional directory entry may hold only a fixed number of characters of the long filename and how many additional directory entries are created is dictated by how many additional directory entries are required to store characters of the long filename which are not already stored in the second directory entry.

16. The computer-readable medium of claim **[14]** *12* wherein the step of creating at least one additional directory entry for the long filename further comprises the step of creating a plurality of additional directory entries.

17. The computer-readable medium of claim **[14]** *12* wherein the step of creating the second directory entry further comprises the step of providing a signature in the second directory entry that identifies that the second directory entry holds the first portion of the long filename.

19. The computer-readable medium of claim **[14]** *12* wherein the step of creating at least one additional directory entry for the long filename further comprises the step of providing a checksum of the first filename in each additional directory entry.

20. In a data processing system having a processor running an operating system and a memory means with memory locations, wherein said memory means stores the

operating system, a computer-readable medium holding computer-executable instructions for performing a method comprising the steps of:

(a) storing in a first of the memory locations of the memory means a first directory entry for a file wherein the first directory entry holds a short filename for the file, said short filename including at most a maximum number of characters that is permissible by an application program;

(b) storing in a second of the memory locations of the memory means that is adjacent to the first of the memory locations a second directory entry for the file wherein the second directory entry holds at least a first portion of a long filename for the file, said long filename including a greater number of characters than the maximum number of characters that is permissible by the application program;

(c) storing at least one additional directory entry in at least one other of the memory locations that is adjacent to the second of the memory locations of the memory means wherein the at least one additional directory entry holds a next portion of the long filename; and

[(c)**]** *(d)* accessing one of the directory entries to locate the file.

21. The computer-readable medium of claim **20** wherein a checksum of the short filename is stored in the second directory entry *and the at least one additional directory entry*.

23. The computer-readable medium of claim **[22]** *20* wherein a signature is stored in *each of the second directory entry and* the *at least one* additional directory entry **[**that**]**, *each signature* uniquely **[**identifies**]** *identifying* which portion of the long filename is stored in **[**the additional**]** *its respective* directory entry.

29. The method as recited in claim 1, wherein the signature is stored at a beginning of the second directory entry and comprises characters that are not permitted to be used as characters of a short filename of a file.

30. The method as recited in claim 5, wherein the signature provided in each additional directory entry is stored at a beginning of that additional directory entry and comprises characters that are not permitted to be used as characters of a short filename of a file.

31. The method as recited in claim 11, wherein the signatures are stored at the beginnings of the second directory entry and the at least one additional directory entry and each signature comprises characters that are not permitted to be used as characters of a short filename of a file.

32. The computer-readable medium as recited in claim 17, wherein the signature provided in the second directory entry is stored at a beginning of the second directory entry and comprises characters that are not permitted to be used as characters of a short filename of a file.

33. The computer-readable medium as recited in claim 18, wherein the signatures provided in each additional directory entry are stored at the beginnings of each additional directory entry and comprise characters that are not permitted to be used as characters of a short filename of a file.

34. The method as recited in claim 23, wherein the signatures stored in the second directory entry and the at least one additional directory entry are stored at a beginning of each respective directory entry and comprise characters that are not permitted to be used as characters of a short filename of a file.

35. The method as recited in claim 24, wherein the step of creating the second directory entry further comprises the step of storing a checksum of the short filename in the second directory entry.

36. The method as recited in claim 24, further comprising the step of creating at least one additional directory entry holding a next portion of the long filename.

37. The method as recited in claim 36 wherein the step of creating at least one additional directory entry further comprises the step of storing a checksum of the short filename in the at least one additional directory entry.

38. The method as recited in claim 36, further comprising storing a signature in each of the second directory entry and the at least one additional directory entry that uniquely identifies which portion of the long filename is stored in that directory entry.

39. The method as recited in claim 38, wherein the signatures stored in the second directory entry and the at least one additional directory entry are stored at a beginning of each respective directory entry and comprise characters that are not permitted to be used as characters of a short filename of a file.

40. The computer-readable medium as recited in claim 28 wherein the step of creating the second directory further comprises the step of storing a checksum of the short filename in the second directory entry.

41. The computer-readable medium as recited in claim 28, further comprising the step of creating at least one additional directory entry holding a next portion of the long filename.

42. The computer-readable medium as recited in claim 41, wherein the step of creating at least one additional directory entry further comprises the step of storing a checksum of the short filename in the at least one additional directory entry.

43. The computer-readable medium as recited in claim 41, further comprising storing a signature in each of the second directory entry and the at least one additional directory entry that uniquely identifies which portion of the long filename is stored that directory entry.

44. The computer-readable medium as recited in claim 43, wherein the signatures stored in the second directory entry and the at least one additional directory entry are stored at a beginning of each respective directory entry and comprise characters that are not permitted to be used as characters of a short filename of a file.

* * * * *

US006621746B1

(12) **United States Patent**
Aasheim et al.

(10) **Patent No.: US 6,621,746 B1**
(45) **Date of Patent: Sep. 16, 2003**

(54) **MONITORING ENTROPIC CONDITIONS OF A FLASH MEMORY DEVICE AS AN INDICATOR FOR INVOKING ERASURE OPERATIONS**

(75) Inventors: **Jered Donald Aasheim**, Bellevue, WA (US); **Yongqi Yang**, Bellevue, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **10/087,097**

(22) Filed: **Feb. 27, 2002**

(51) **Int. Cl.**[7] **G11C 7/00**

(52) **U.S. Cl.** **365/185.29**; 365/185.33; 365/218; 711/103

(58) **Field of Search** 365/185.29, 185.33, 365/218, 230.03; 711/103

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,530,828 A	*	6/1996	Kaki et al.	711/103
5,598,370 A	*	1/1997	Niijima et al.	365/185.33
5,734,816 A	*	3/1998	Niijima et al.	714/8
5,745,418 A		4/1998	Ma et al.	
5,867,641 A		2/1999	Jenett	
5,887,198 A		3/1999	Houlberg et al.	
5,937,425 A		8/1999	Ban	
5,956,473 A		9/1999	Ma et al.	
6,014,724 A		1/2000	Jenett	
6,279,069 B1		8/2001	Robinson et al.	
6,347,051 B2	*	2/2002	Yamagami et al.	365/185.09

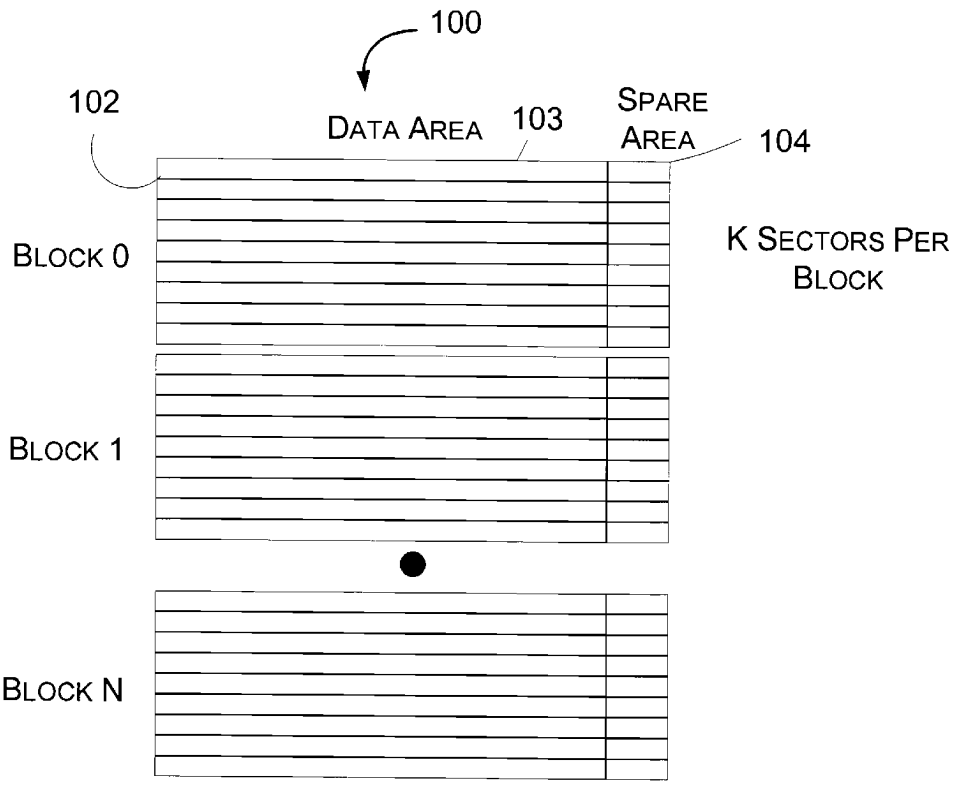
* cited by examiner

Primary Examiner—Son T. Dinh
(74) *Attorney, Agent, or Firm*—Lee & Hayes, PLLC

(57) **ABSTRACT**

Erase operations are performed on a flash memory device by monitoring the entropic nature of the flash memory device. In one implementation, flash abstraction logic, tracks how many physical sectors are free to receive data; track how many physical sectors contain data that is dirty, and compare whether the physical sectors that are free to receive data outnumber the physical sectors that contain data that is dirty. A compactor performs an erase operation of one or more blocks when the physical sectors that contain data that is dirty outnumber the physical sectors that are free to receive data. In another implementation, the flash abstraction logic tracks how many physical sector addresses are free to receive data, and track when the physical sector addresses that are free to receive data are insufficient in quantity to receive write requests from a file system. The compactor executes an erase operation of one or more blocks if the physical sector addresses that are free to receive data are insufficient in quantity.

26 Claims, 15 Drawing Sheets



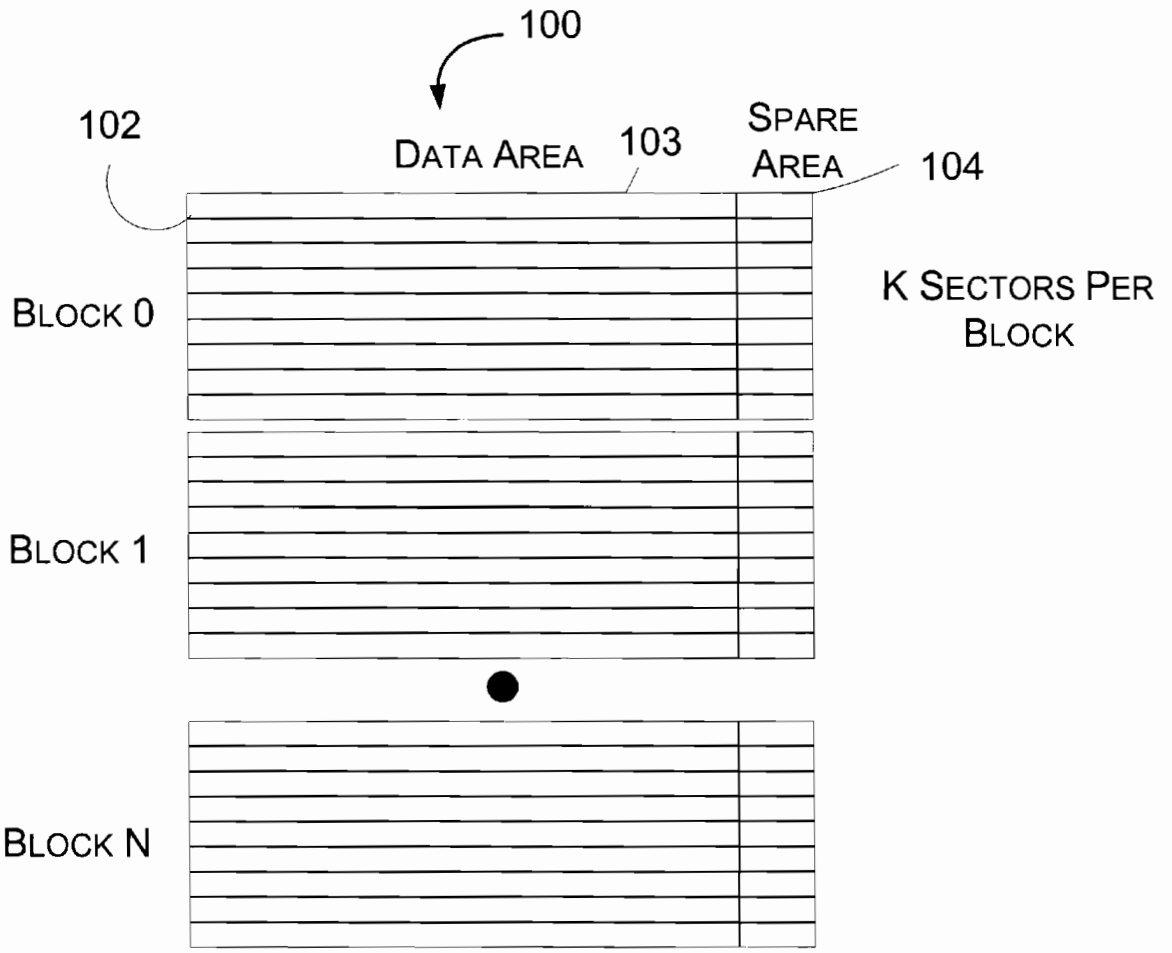

100
102
DATA AREA
103
SPARE AREA
104
BLOCK 0
K SECTORS PER BLOCK
BLOCK 1
BLOCK N

Fig. 1

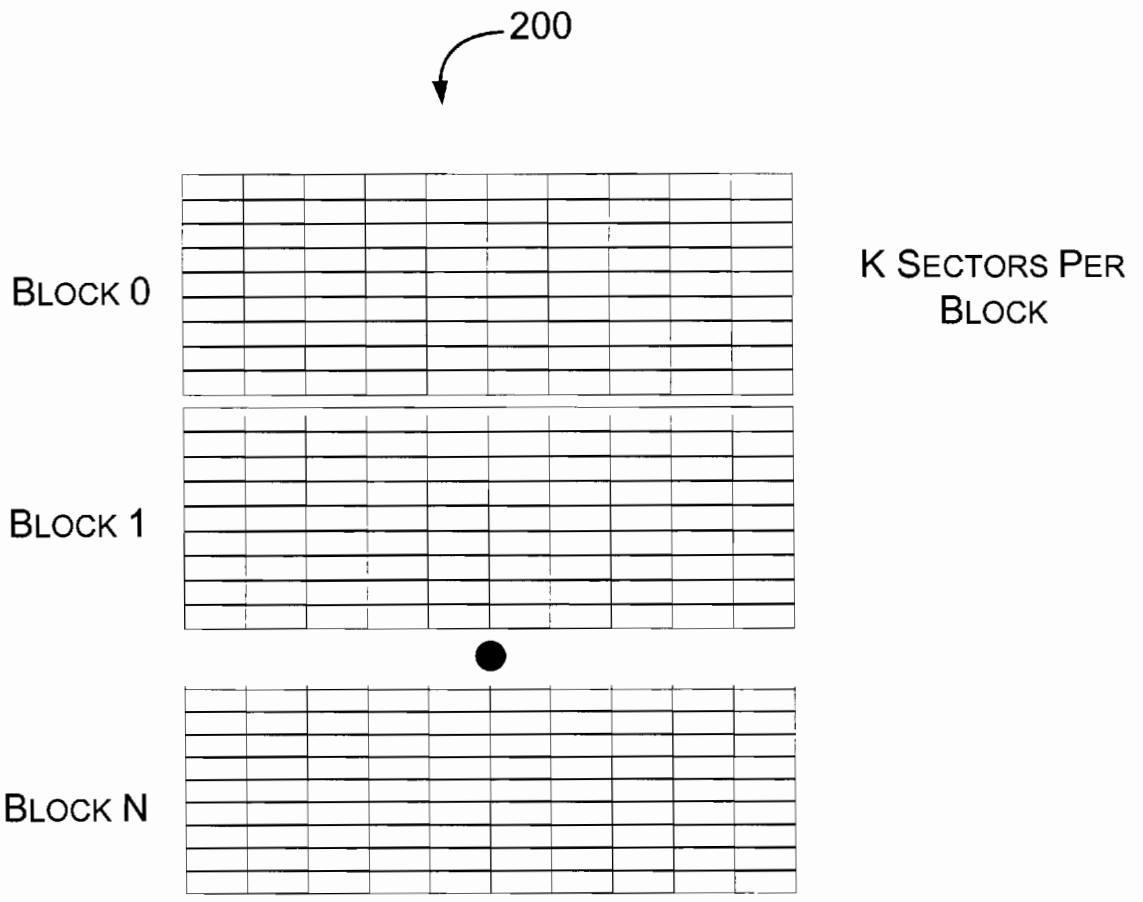


Fig. 2

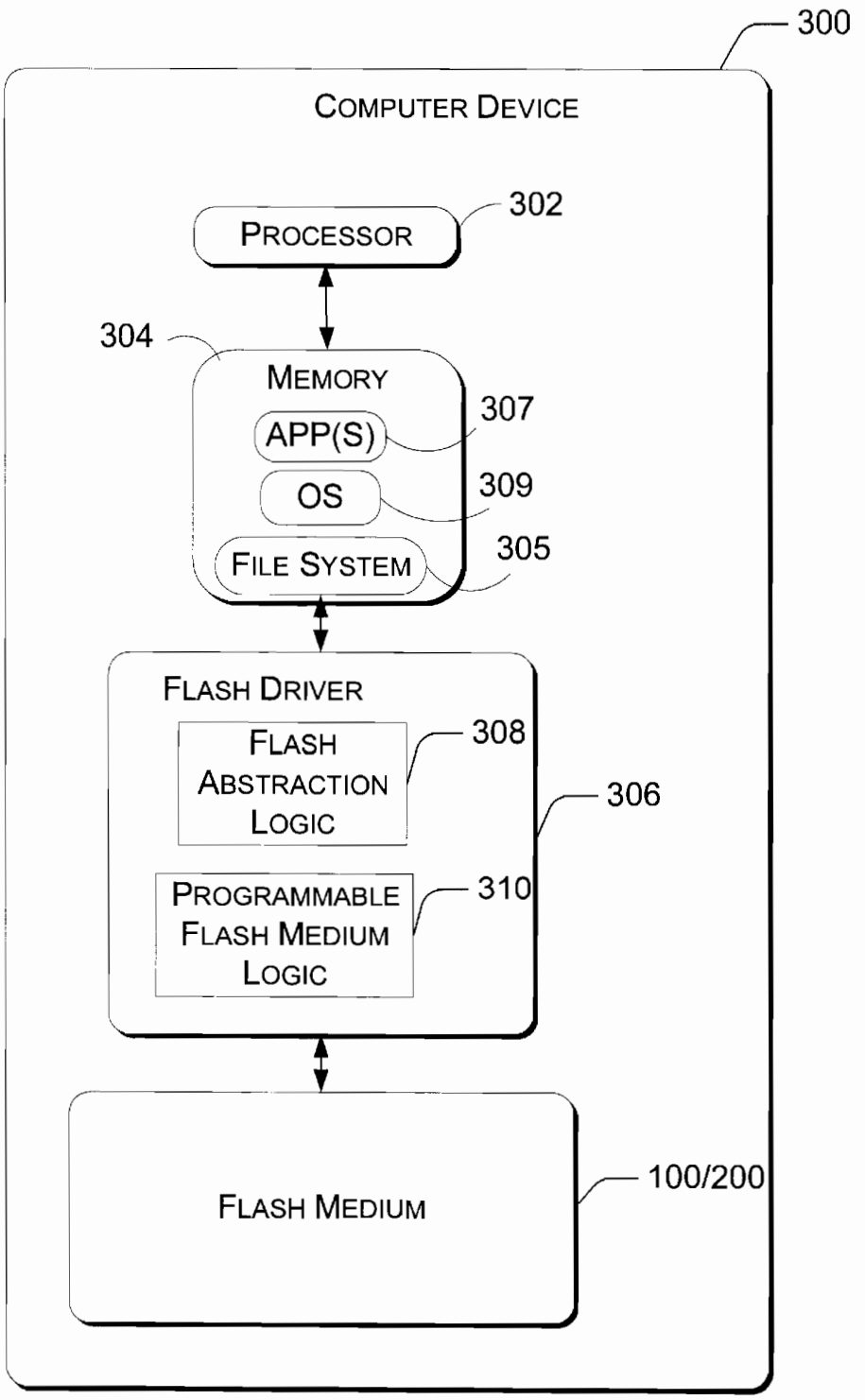

300
COMPUTER DEVICE
PROCESSOR
302
304
MEMORY
APP(S)
307
OS
309
FILE SYSTEM
305
FLASH DRIVER
FLASH ABSTRACTION LOGIC
308
306
PROGRAMMABLE FLASH MEDIUM LOGIC
310
FLASH MEDIUM
100/200

Fig. 3

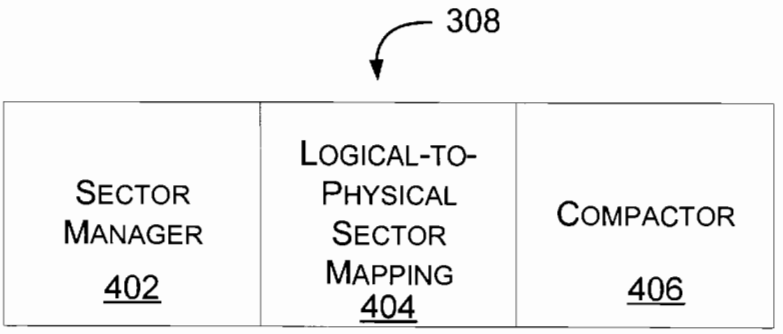


Fig. 4

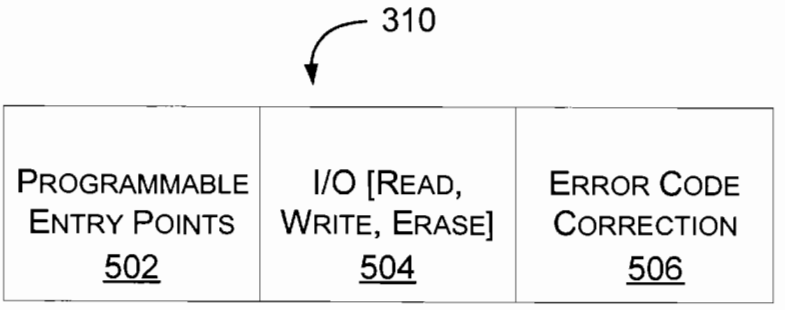


Fig. 5

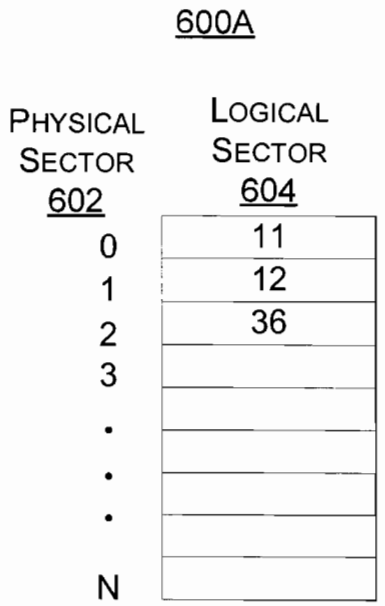


Fig. 6A

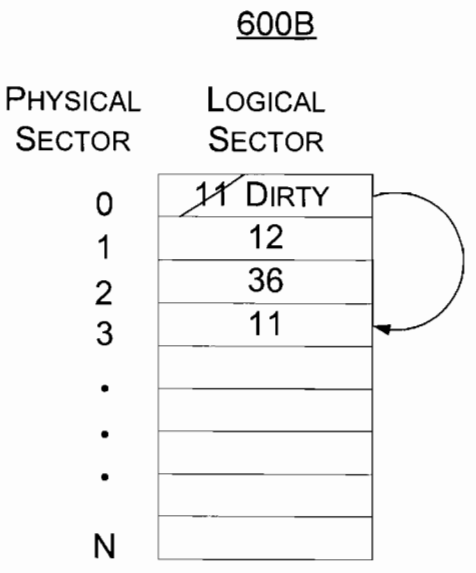


Fig. 6B

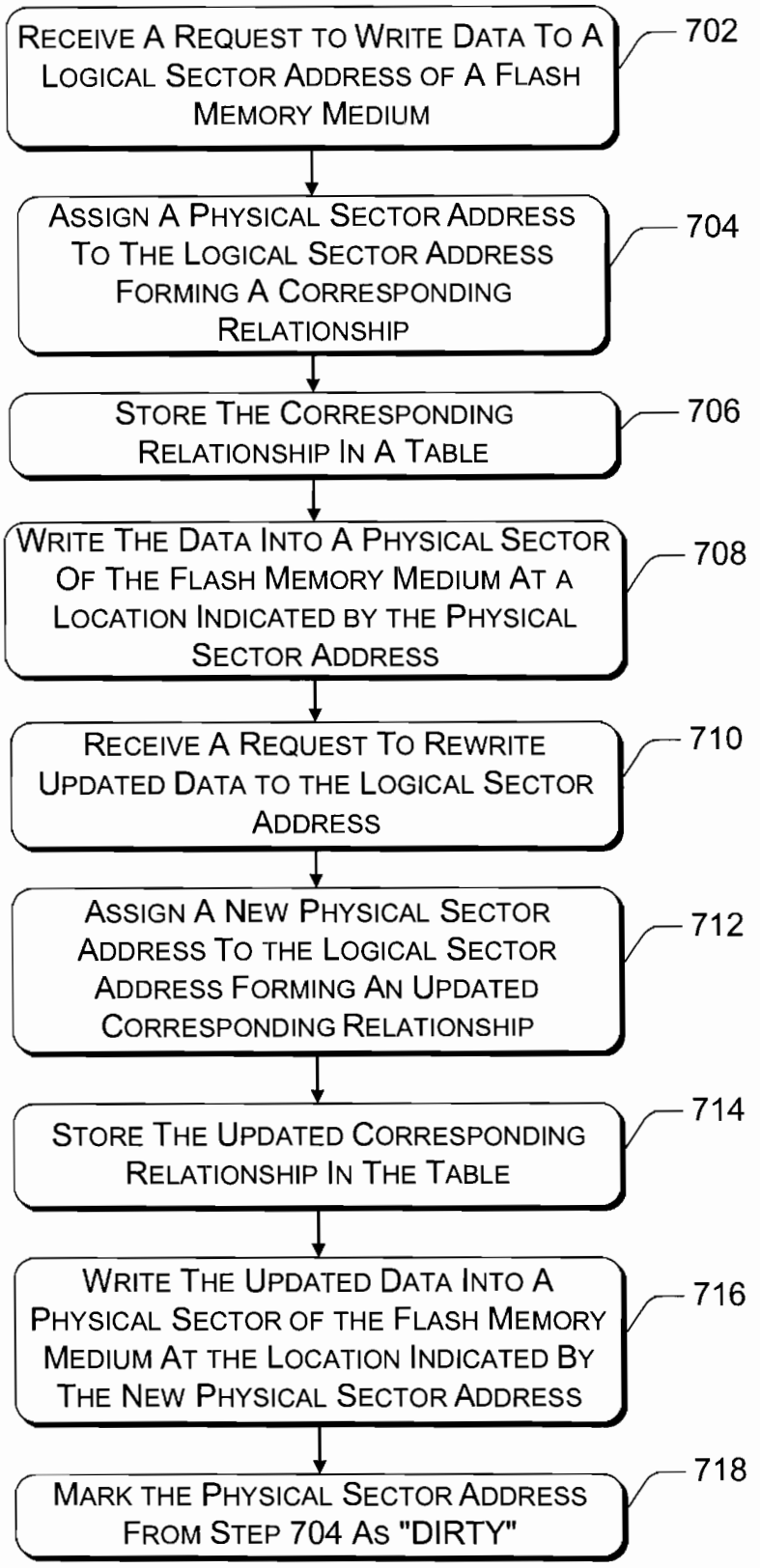

700
RECEIVE A REQUEST TO WRITE DATA TO A LOGICAL SECTOR ADDRESS OF A FLASH MEMORY MEDIUM
702
ASSIGN A PHYSICAL SECTOR ADDRESS TO THE LOGICAL SECTOR ADDRESS FORMING A CORRESPONDING RELATIONSHIP
704
STORE THE CORRESPONDING RELATIONSHIP IN A TABLE
706
WRITE THE DATA INTO A PHYSICAL SECTOR OF THE FLASH MEMORY MEDIUM AT A LOCATION INDICATED BY THE PHYSICAL SECTOR ADDRESS
708
RECEIVE A REQUEST TO REWRITE UPDATED DATA TO THE LOGICAL SECTOR ADDRESS
710
ASSIGN A NEW PHYSICAL SECTOR ADDRESS TO THE LOGICAL SECTOR ADDRESS FORMING AN UPDATED CORRESPONDING RELATIONSHIP
712
STORE THE UPDATED CORRESPONDING RELATIONSHIP IN THE TABLE
714
WRITE THE UPDATED DATA INTO A PHYSICAL SECTOR OF THE FLASH MEMORY MEDIUM AT THE LOCATION INDICATED BY THE NEW PHYSICAL SECTOR ADDRESS
716
MARK THE PHYSICAL SECTOR ADDRESS FROM STEP 704 AS "DIRTY"
718

Fig. 7

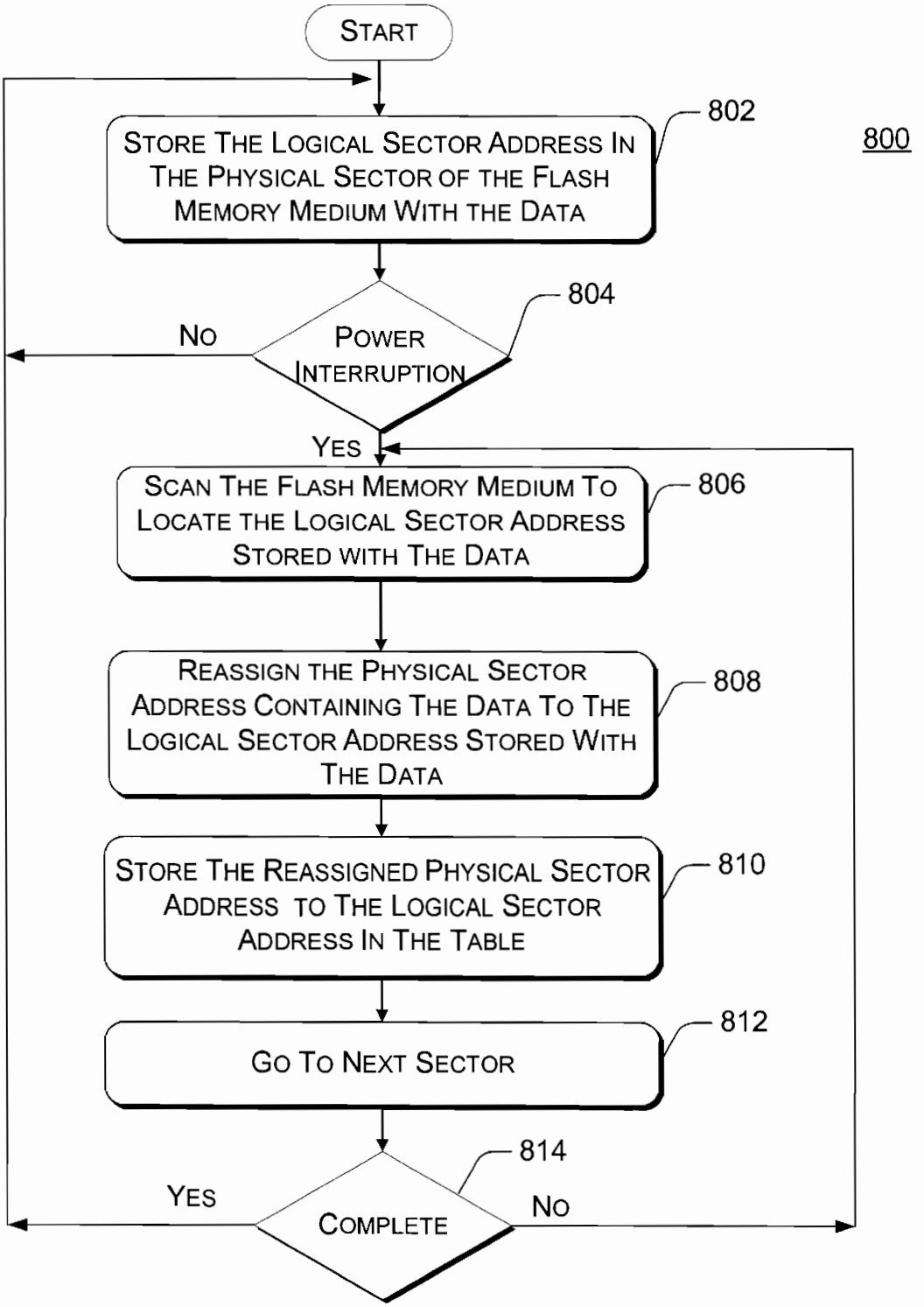

START
STORE THE LOGICAL SECTOR ADDRESS IN THE PHYSICAL SECTOR OF THE FLASH MEMORY MEDIUM WITH THE DATA
802
800
POWER INTERRUPTION
804
NO
YES
SCAN THE FLASH MEMORY MEDIUM TO LOCATE THE LOGICAL SECTOR ADDRESS STORED WITH THE DATA
806
REASSIGN THE PHYSICAL SECTOR ADDRESS CONTAINING THE DATA TO THE LOGICAL SECTOR ADDRESS STORED WITH THE DATA
808
STORE THE REASSIGNED PHYSICAL SECTOR ADDRESS TO THE LOGICAL SECTOR ADDRESS IN THE TABLE
810
GO TO NEXT SECTOR
812
COMPLETE
814
YES
NO

Fig. 8

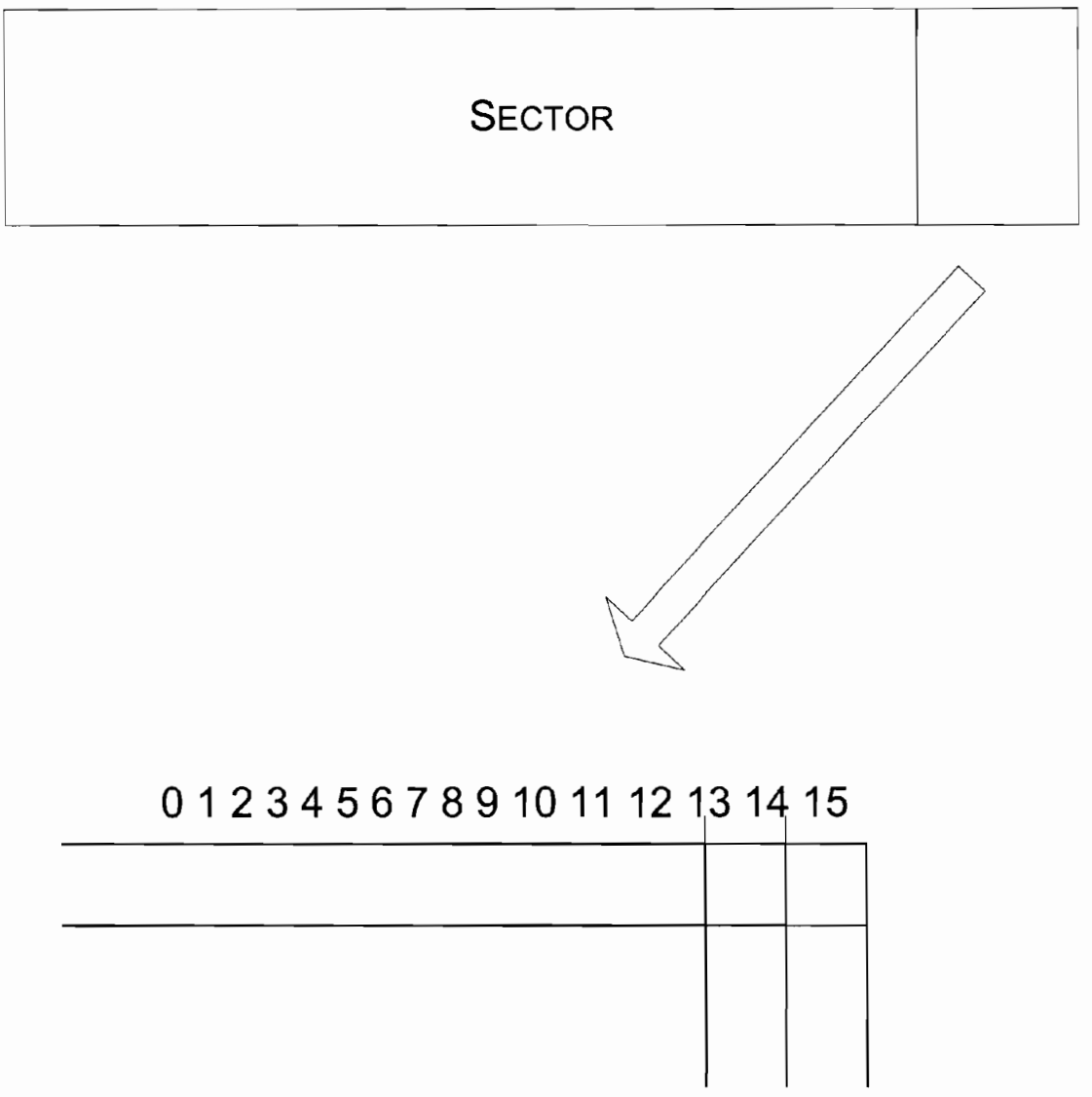


Fig. 9

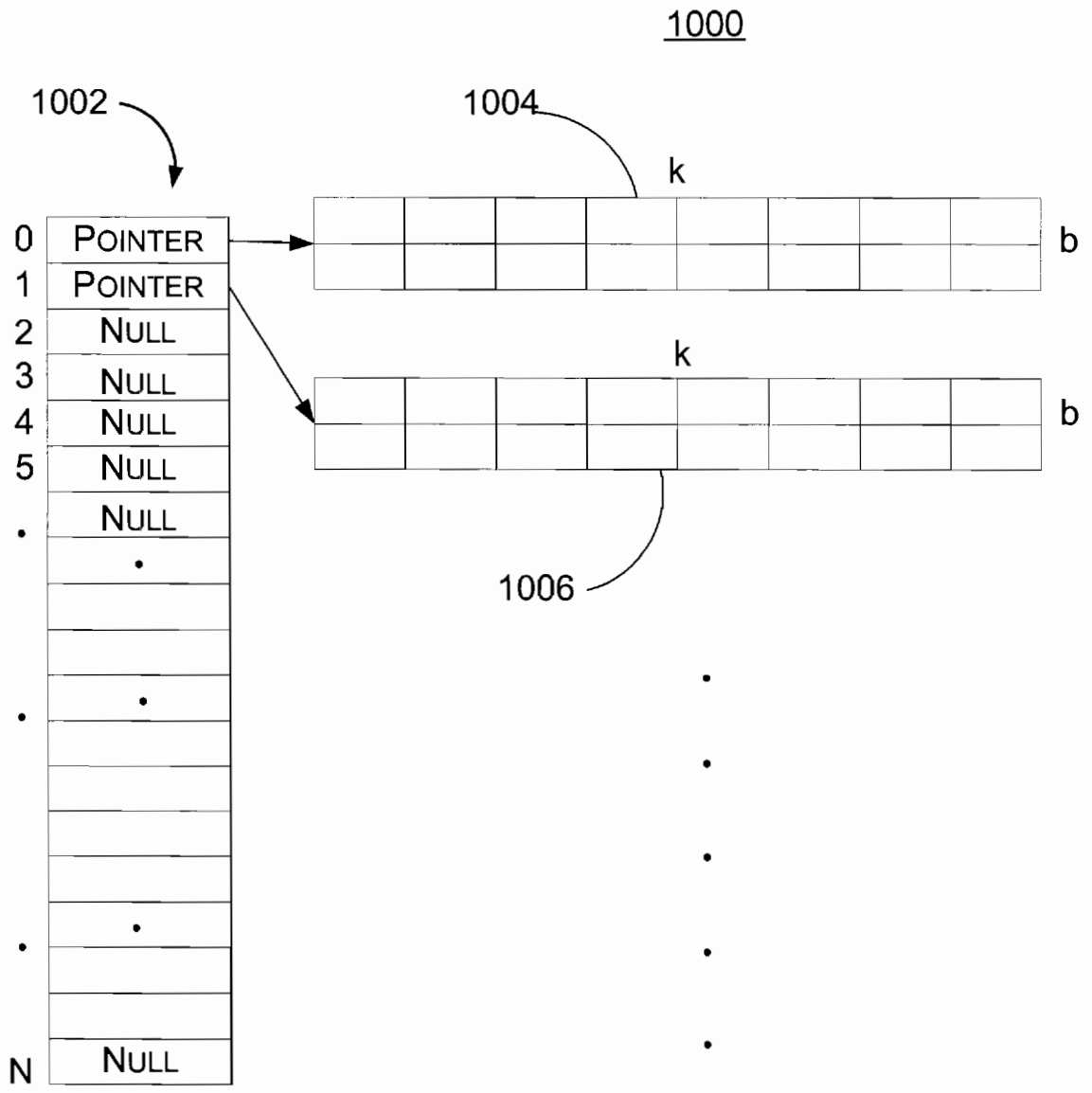

1000
1002
1004
k
0
POINTER
1
POINTER
2
NULL
3
NULL
4
NULL
5
NULL
NULL
b
k
b
1006
N
NULL

Fig. 10

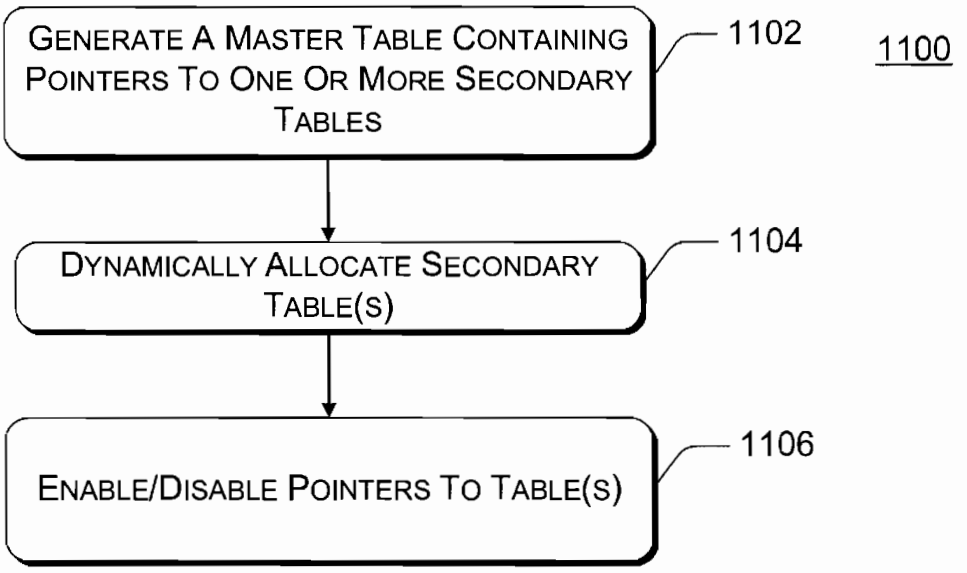

GENERATE A MASTER TABLE CONTAINING POINTERS TO ONE OR MORE SECONDARY TABLES
1102
1100
DYNAMICALLY ALLOCATE SECONDARY TABLE(S)
1104
ENABLE/DISABLE POINTERS TO TABLE(S)
1106

Fig. 11

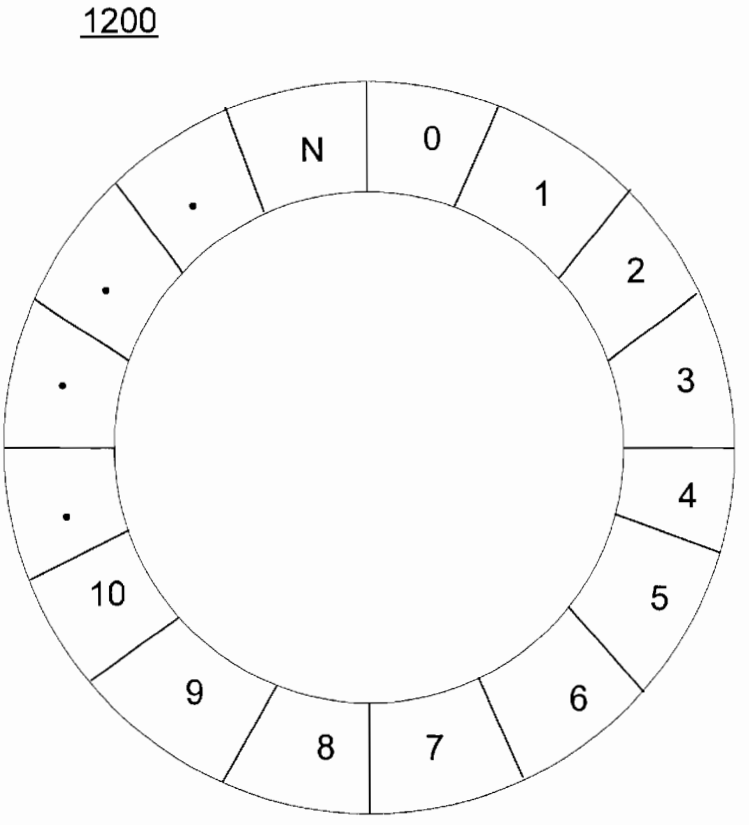

1200
N
0
1
2
3
4
5
6
7
8
9
10

Fig. 12

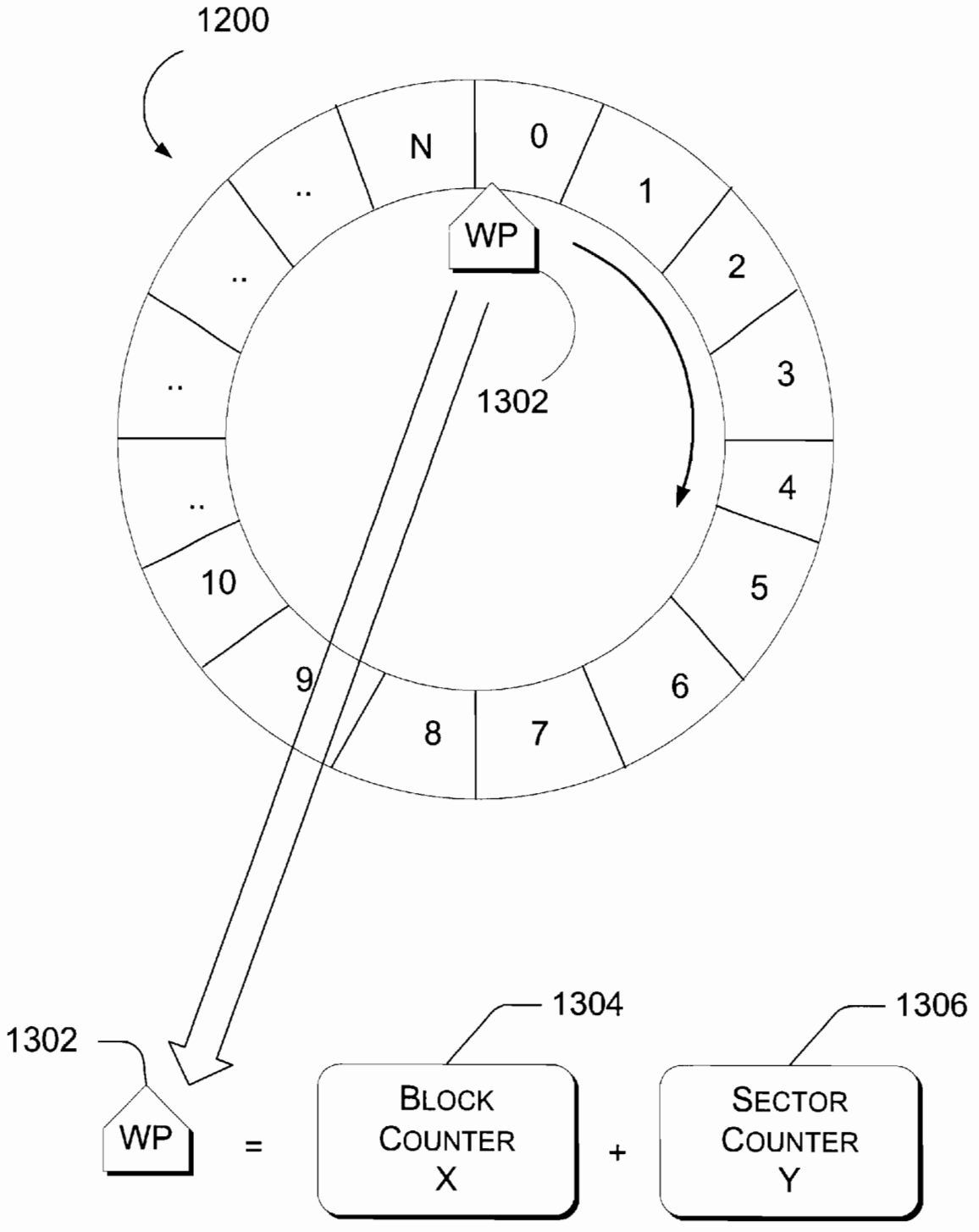

1200
N
0
1
2
3
4
5
6
7
8
9
10
..
..
..
..
WP
1302
1302
WP
=
1304
BLOCK COUNTER X
+
1306
SECTOR COUNTER Y

Fig. 13

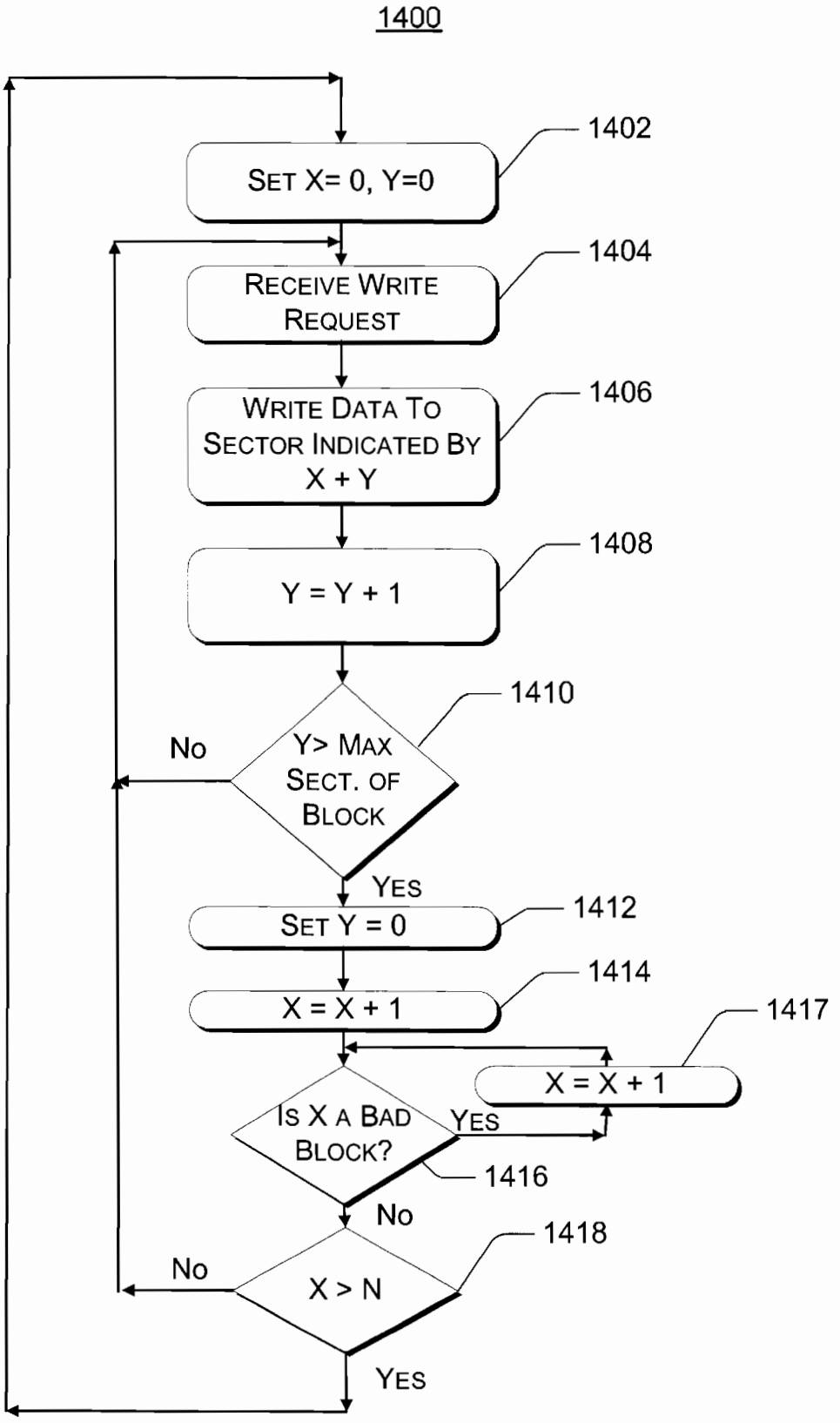

1400
1402
SET X= 0, Y=0
1404
RECEIVE WRITE REQUEST
1406
WRITE DATA TO SECTOR INDICATED BY X + Y
1408
Y = Y + 1
1410
Y> MAX SECT. OF BLOCK
No
YES
1412
SET Y = 0
1414
X = X + 1
1417
X = X + 1
IS X A BAD BLOCK?
YES
1416
No
1418
X > N
No
YES

Fig. 14

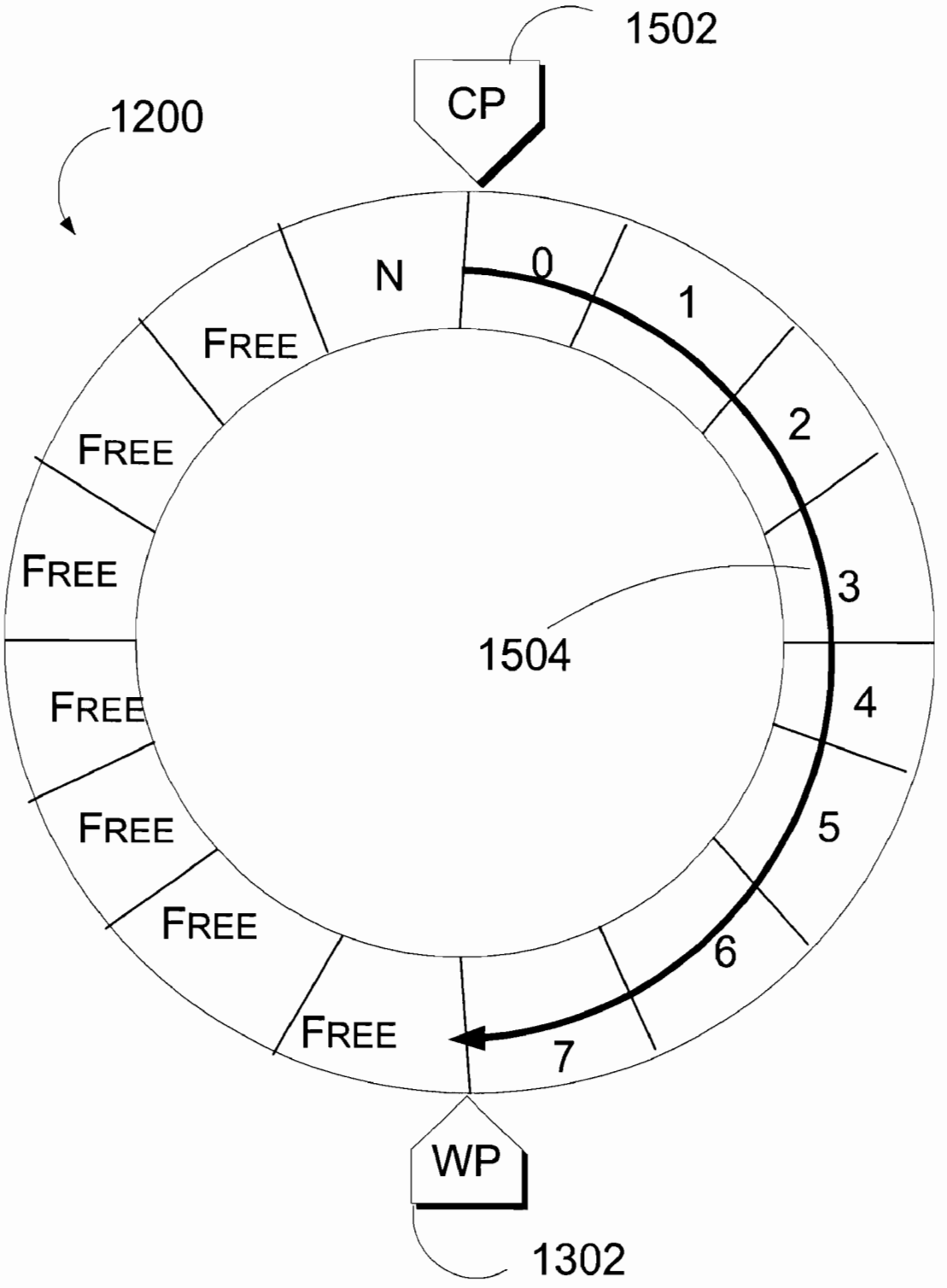

1502
CP
1200
N
0
1
2
3
4
5
6
7
FREE
FREE
FREE
FREE
FREE
FREE
FREE
1504
WP
1302

Fig. 15

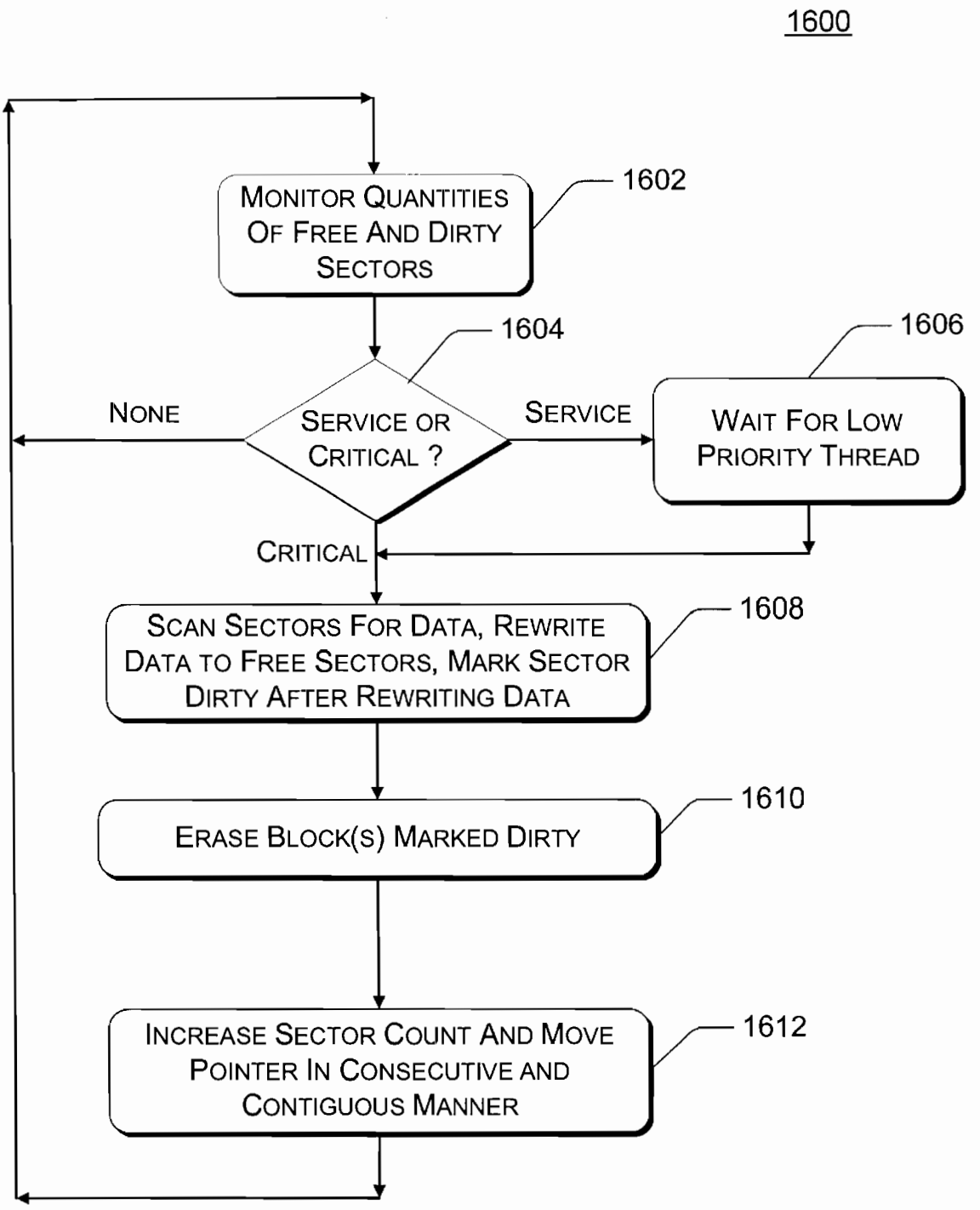

1600
MONITOR QUANTITIES OF FREE AND DIRTY SECTORS
1602
1604
SERVICE OR CRITICAL ?
NONE
SERVICE
1606
WAIT FOR LOW PRIORITY THREAD
CRITICAL
SCAN SECTORS FOR DATA, REWRITE DATA TO FREE SECTORS, MARK SECTOR DIRTY AFTER REWRITING DATA
1608
ERASE BLOCK(S) MARKED DIRTY
1610
INCREASE SECTOR COUNT AND MOVE POINTER IN CONSECUTIVE AND CONTIGUOUS MANNER
1612

Fig. 16

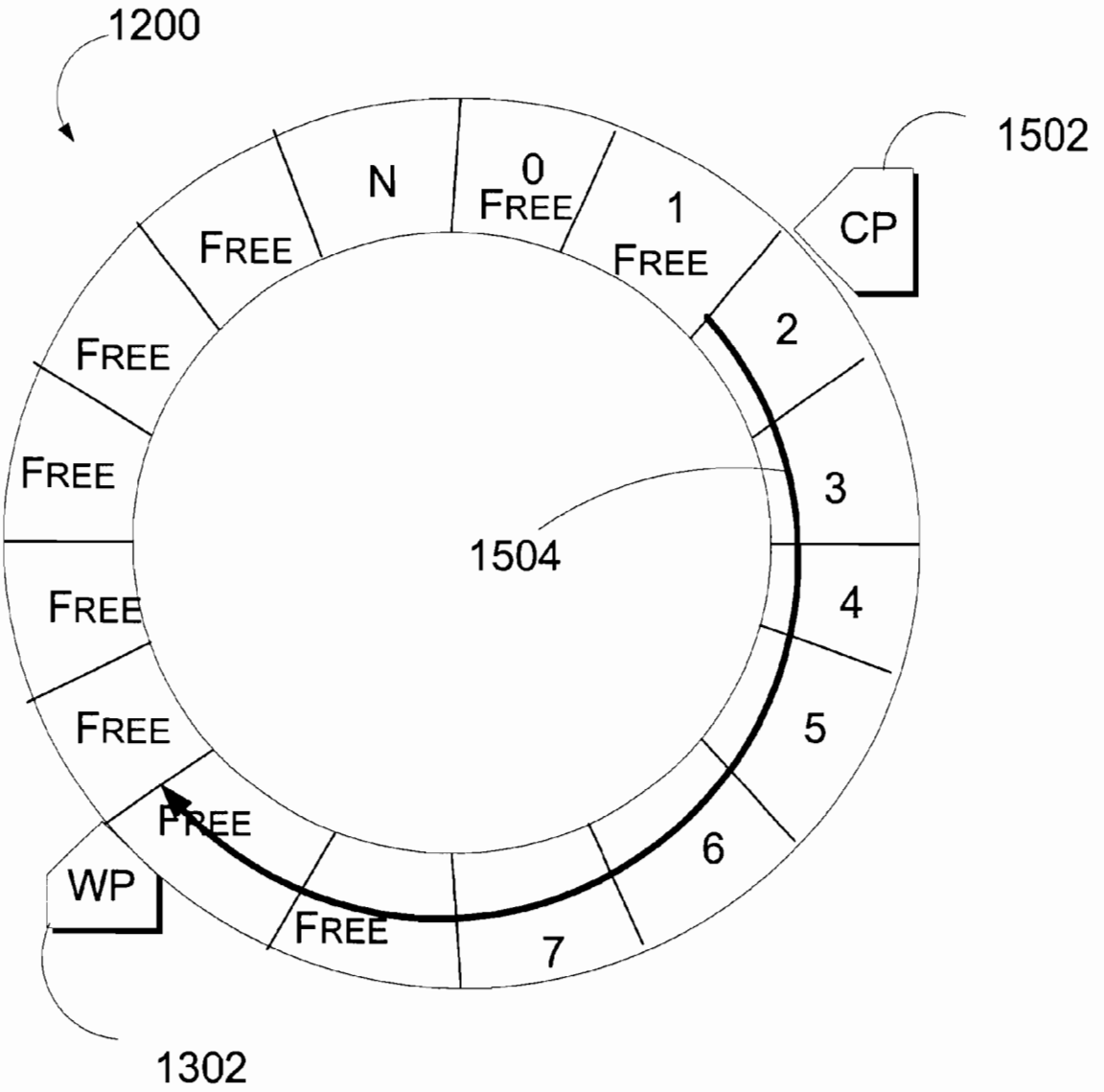

1200
1502
N
0
FREE
1
FREE
CP
FREE
2
FREE
3
FREE
1504
4
FREE
5
FREE
FREE
6
WP
FREE
7
1302

Fig. 17

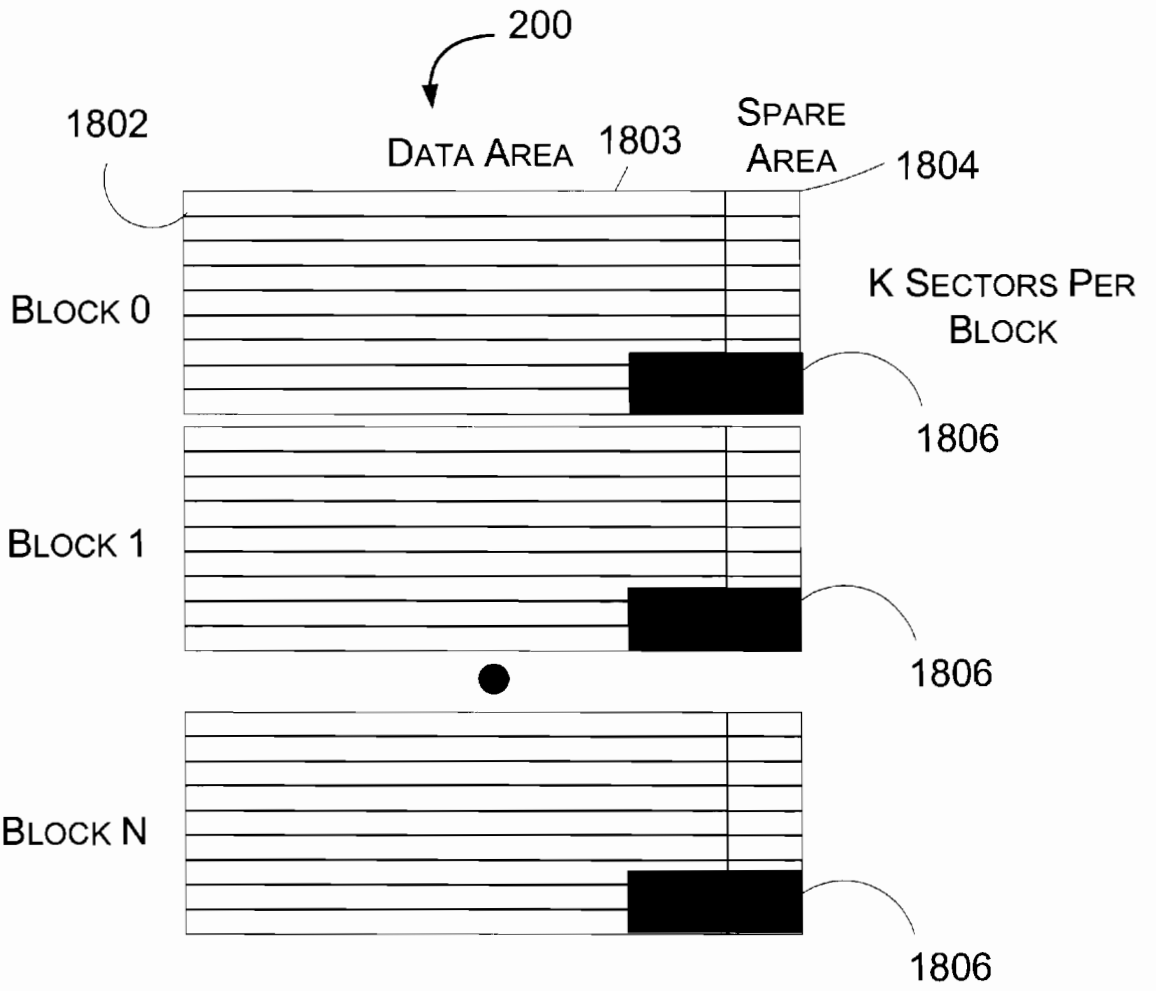

200
1802
DATA AREA
1803
SPARE AREA
1804
BLOCK 0
K SECTORS PER BLOCK
1806
BLOCK 1
1806
BLOCK N
1806

Fig. 18

MONITORING ENTROPIC CONDITIONS OF A FLASH MEMORY DEVICE AS AN INDICATOR FOR INVOKING ERASURE OPERATIONS

TECHNICAL FIELD

This invention relates to flash memory devices, and more particularly, monitoring when to perform an erase operation in a flash memory device.

BACKGROUND

Flash memory devices have many advantages for a large number of applications. These advantages include their non-volatility, speed, ease of erasure and reprogramming, small physical size and related factors. There are no mechanical moving parts and as a result such systems are not subject to failures of the type most often encountered with hard disk storage systems.

Flash memory devices have many characteristics that are different from other memory devices. One major difference is that a block containing existing data in flash memory devices cannot be overwritten with new data. Existing data must be completely erased (also referred to as "cleaned") from a block before data can be written into memory locations again. Blocks can only be erased a limited number of times before the flash memory device becomes unusable.

Additionally, most block erasures stalls other operations from occurring such as read and write operations to the flash memory device. Most flash memory systems attempt, therefore, to minimize erasures to specific times, such as at initialization or powering-off of a computer system; or at periodically scheduled times.

The problem with periodic scheduled erasure times is that they may occur more often than actually needed which in turn will prematurely shorten the life of a flash memory device.

The problem with scheduling erasures at initialization or powering off of a device is that they may also occur more often than needed which in turn will prematurely shorten the life of a flash memory device. They may also not occur enough in which case the flash memory device may be become full and not accept new data.

SUMMARY

A system and method invokes an erase operation in a flash memory device by monitoring the entropic nature of the flash memory device. In one implementation, flash abstraction logic, tracks how many physical sectors are free to receive data; track how many physical sectors contain data that is dirty, and compare whether the physical sectors that are free to receive data outnumber the physical sectors that contain data that is dirty. A compactor performs an erase operation of one or more blocks when the physical sectors that contain data that is dirty outnumber the physical sectors that are free to receive data.

In another implementation, the flash abstraction logic tracks how many physical sector addresses are free to receive data, and track when the physical sector addresses that are free to receive data are insufficient in quantity to receive write requests from a file system. The compactor executes an erase operation of one or more blocks if the physical sector addresses that are free to receive data are insufficient in quantity.

BRIEF DESCRIPTION OF THE DRAWINGS

The detailed description is described with reference to the accompanying figures. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears.

FIG. **1** illustrates a logical representation of a NAND flash memory medium.

FIG. **2** illustrates a logical representation of a NOR flash memory medium.

FIG. **3** illustrates pertinent components of a computer device, which uses one or more flash memory devices to store information.

FIG. **4** illustrates a block diagram of flash abstraction logic.

FIG. **5** illustrates an exemplary block diagram of a flash medium logic.

FIG. **6**A shows a data structure used to store a corresponding relationship between logical sector addresses and physical sector addresses.

FIG. **6**B shows a data structure which is the same as the data structure in FIG. **6**B, except its contents have been updated.

FIG. **7** illustrates a process used to track data on the flash memory medium when the file system issues write requests to the flash driver.

FIG. **8** illustrates a process for safeguarding mapping of logical-to-physical sector address information stored in volatile data structures, such as the data structures shown in FIGS. **6**A and **6**B.

FIG. **9** illustrates a location within the flash memory medium in which the logical sector address can be stored for safeguarding in the event of a power failure.

FIG. **10** illustrates a dynamic look-up data structure to track data stored in the flash memory medium.

FIG. **11** illustrates a process for dynamically allocating look-up data structures for tracking data on the flash memory medium.

FIG. **12** is a diagram of the flash memory medium viewed and/or treated as a continuous circle by the flash driver.

FIG. **13** depicts another illustration of the media viewed as a continuous circle.

FIG. **14** illustrates a process used by the sector manager to determine the next available free sector location for the flash driver to store data on the medium.

FIG. **15** illustrates another view of media treated as a continuous circle.

FIG. **16** is a flow chart illustrating a process used by the compactor to recycle sectors.

FIG. **17** shows one exemplary result from the process illustrated in FIG. **16**.

FIG. **18** illustrates a logical representation of a NOR flash memory medium divided in way to better support the processes and techniques implemented by the flash driver.

DETAILED DESCRIPTION

The following discussion is directed to flash drivers. The subject matter is described with specificity to meet statutory requirements. However, the description itself is not intended to limit the scope of this patent. Rather, the inventors have contemplated that the claimed subject matter might also be embodied in other ways, to include different elements or combinations of elements similar to the ones described in this document, in conjunction with other present or future technologies.

Overview

This discussion assumes that the reader is familiar with basic operating principles of flash memory media.

Nevertheless, a general introduction to two common types of nonvolatile random access memory, NAND and NOR Flash memory media, is provided to better understand the exemplary implementations described herein. These two example flash memory media were selected for their current popularity, but their description is not intended to limit the described implementations to these types of flash media. Other electrically erasable and programmable read-only memories (EEPROMs) would work too. In most examples used throughout this Detailed Description numbers shown in data structures are in decimal format for illustrative purposes.

Universal Flash Medium Operating Characteristics

FIG. **1** and FIG. **2** illustrate logical representations of example NAND and NOR flash memory media **100**, **200**, respectively. Both media have universal operating characteristics that are common to each, respectively, regardless of the manufacturer. For example referring to FIG. **1**, a NAND flash memory medium is generally split into contiguous blocks (**0**, **1**, through N). Each block **0**, **1**, **2**, etc. is further subdivided into K sectors **102**; standard commercial NAND flash media commonly contain 8, 16, or 32 sectors per block. The amount of blocks and sectors can vary, however, depending on the manufacturer. Some manufacturers refer to "sectors" as "pages." Both terms as used herein are equivalent and interchangeable.

Each sector **102** is further divided into two distinct sections, a data area **103** used to store information and a spare area **104** which is used to store extra information such as error correction code (ECC). The data area **103** size is commonly implemented as 512 bytes, but again could be more or less depending on the manufacturer. At 512 bytes, the flash memory medium allows most file systems to treat the medium as a nonvolatile memory device, such as a fixed disk (hard drive). As used herein RAM refers generally to the random access memory family of memory devices such as DRAM, SRAM, VRAM, VDO, and so forth. Commonly, the size of the area spare **104** is implemented as 16 bytes of extra storage for NAND flash media devices. Again, other sizes, greater or smaller can be selected. In most instances, the spare area **104** is used for error correcting codes, and status information.

A NOR memory medium **200** is different than NAND memory medium in that blocks are not subdivided into physical sectors. Similar to RAM, each byte stored within a block of NOR memory medium is individually addressable. Practically, however, blocks on NOR memory medium can logically be subdivided into physical sectors with the accompanying spare area.

Aside from the overall layout and operational comparisons, some universal electrical characteristics (also referred to herein as "memory requirements" or "rules") of flash devices can be summarized as follows:

1. Write operations to a sector can change an individual bit from a logical '1' to a logical '0', but not from a logical '0' to logical '1' (except for case No. 2 below);
2. Erasing a block sets all of the bits in the block to a logical '1';
3. It is not generally possible to erase individual sectors/bytes/bits in a block without erasing all sectors/bytes within the same block;
4. Blocks have a limited erase lifetime of between approximately 100,000 to 1,000, 000 cycles;
5. NAND flash memory devices use ECC to safeguard against data corruption due to leakage currents; and
6. Read operations do not count against the write/erase lifetime.

Flash Driver Architecture

FIG. **3** illustrates pertinent components of a computer device **300**, which uses one or more flash memory devices to store information. Generally, various different general purpose or special purpose computing system configurations can be used for computer device **300**, including but not limited to personal computers, server computers, hand-held or laptop devices, portable communication devices, multiprocessor systems, microprocessor systems, microprocessor-based systems, programmable consumer electronics, gaming systems, multimedia systems, the combination of any of the above example devices and/or systems, and the like.

Computer device **300** generally includes a processor **302**, memory **304**, and a flash memory media **100**/**200**. The computer device **300** can include more than one of any of the aforementioned elements. Other elements such as power supplies, keyboards, touch pads, I/O interfaces, displays, LEDs, audio generators, vibrating devices, and so forth are not shown, but could easily be a part of the exemplary computer device **300**.

Memory **304** generally includes both volatile memory (e.g., RAM) and non-volatile memory (e.g., ROM, PCMCIA cards, etc.). In most implementations described below, memory **304** is used as part of computer device's **302** cache, permitting application data to be accessed quickly without having to permanently store data on a non-volatile memory such as flash medium **100**/**200**.

An operating system **309** is resident in the memory **304** and executes on the processor **302**. An example operating system implementation includes the Windows® CE operating system from Microsoft Corporation, but other operation systems can be selected from one of many operating systems, such as DOS, UNIX, etc. For purposes of illustration, programs and other executable program components such as the operating system are illustrated herein as discrete blocks, although it is recognized that such programs and components reside at various times in different storage components of the computer, and are executed by the processor(s) of the computer device **300**.

One or more application programs **307** are loaded into memory **304** and run on the operating system **309**. Examples of applications include, but are not limited to, email programs, word processing programs, spreadsheets programs, Internet browser programs, as so forth.

Also loaded into memory **304** is a file system **305** that also runs on the operating system **309**. The file system **305** is generally responsible for managing the storage and retrieval of data to memory devices, such as magnetic hard drives, and this exemplary implementation flash memory media **100**/**200**. Most file systems **305** access and store information at a logical level in accordance with the conventions of the operating system the file system **305** is running. It is possible for the file system **305** to be part of the operating system **309** or embedded as code as a separate logical module.

Flash driver **306** is implemented to function as a direct interface between the file system **305** and flash medium **100**/**200**. Flash driver **306** enables computer device **300** through the file system **305** to control flash medium **100**/**200** and ultimately send/retrieve data. As shall be described in more detail, however, flash driver **306** is responsible for more than read/write operations. Flash driver **306** is implemented to maintain data integrity, perform wear-leveling of the flash medium, minimize data loss during a power interruption to computer device **300** and permit OEMs of computer devices **300** to support their respective flash memory devices regardless of the manufacturer. The flash driver **306**

is file system agnostic. That means that the flash driver **306** supports many different types of files systems, such as File Allocation Data structure File System (FAT16), (FAT32), and other file systems. Additionally, flash driver **306** is flash memory medium agnostic, which likewise means driver **306** supports flash memory devices regardless of the manufacturer of the flash memory device. That is, the flash driver **306** has the ability to read/write/erase data on a flash medium and can support most, if not all, flash devices.

In the exemplary implementation, flash driver **306** resides as a component within operating system **309**, that when executed serves as a logical interface module between the file system **305** and flash medium **100/200**. The flash driver **306** is illustrated as a separate box **306** for purposes of demonstrating that the flash driver when implemented serves as an interface. Nevertheless, flash driver **306** can reside in other applications, part of the file system **305** or independently as separate code on a computer-readable medium that executes in conjunction with a hardware/firmware device.

In one implementation, flash driver **306** includes: a flash abstraction logic **308** and a programmable flash medium logic **310**. Flash abstraction logic **308** and programmable medium logic **310** are coded instructions that support various features performed by the flash driver **306**. Although the exemplary implementation is shown to include these two elements, various features from each of the flash abstraction logic **308** and flash medium logic **310** may be selected to carry out some of the more specific implementations described below. So while the described implementation shows two distinct layers of logic **308/310**, many of the techniques described below can be implemented without necessarily requiring all or a portion of the features from either layer of logic. Furthermore, the techniques may be implemented without having the exact division of responsibilities as described below.

In one implementation, the Flash abstraction logic **308** manages those operating characteristics that are universally common to flash memory media. These universal memory requirements include wear-leveling, maintaining data integrity, and handling recovery of data after a power failure. Additionally, the flash abstraction logic **308** is responsible for mapping information stored at a physical sector domain on the flash memory medium **100/200** to a logical sector domain associated with the file system **305**. That is, the flash abstraction logic **308** tracks data going from a logical-to-physical sector addresses and/or from a physical-to-logical sector addresses. Driver **306** uses logical-to-physical sector addresses for both read/write operations. Driver **306** goes from physical-to-logical sector addresses when creating a look-up table (to be described below) during driver initialization. Some of the more specific commands issued by the file system that are dependent upon a certain type of flash memory media are sent directly to the flash medium logic **310** for execution and translation. Thus, the flash abstraction logic **308** serves as a manager to those universal operations, which are common to flash memory media regardless of the manufacturer for the media, such as wear-leveling, maintaining data integrity, handling data recovery after a power failure and so forth.

FIG. **4** illustrates an exemplary block diagram of the flash abstraction logic **308**. Flash abstraction logic **308** includes a sector manager **402**, a logical-to-physical sector mapping module **404**, and a compactor **406**. Briefly, the sector manager **402** provides a pointer to a sector available, i.e., "free" to receive new data. The logical-to-physical sector mapping module **404** manages data as it goes from a file system domain of logical sector addressing to a flash medium domain of physical sector addressing. The compactor **406** provides a mechanism for clearing blocks of data (also commonly referred to in the industry as "erasing") to ensure that enough free sectors are available for writing data. Additionally, the compactor **406** helps the driver **306** system perform uniform and even wear leveling. All these elements shall be described in more detail below.

Referring back to FIG. **3**, the flash medium logic **310** is used to translate logical commands, received from either the flash abstraction logic **308** or file system **305**, to physical sector commands for issuance to the flash memory medium **100/200**. For instance, the flash medium logic **310** reads, writes, and erases data to and/or from the flash memory medium. The flash medium logic **310** is also responsible for performing ECC (if necessary). In one implementation, the flash medium logic **310** is programmable to permit users to match particular flash medium requirements of a specific manufacturer. Thus, the flash medium logic **310** is configured to handle specific nuances, ECC, and specific commands associated with controlling physical aspects of flash medium **100/200**.

FIG. **5** illustrates an exemplary block diagram of the flash medium logic **310**. As shown, the flash medium logic **310** includes a programmable entry point module **502**, I/O module **504** and an ECC module **506**. The programmable entry point module **502** defines a set of programming interfaces to communicate between flash abstraction logic **308** and flash medium **100/200**. In other words, the programmable entry points permit manufacturers of computer devices **300** to program the flash media logic **310** to interface with the actual flash memory medium **100/200** used in the computer device **300**. The I/O module **504** contains specific code necessary for read/write/erase commands that are sent to the Flash memory medium **100/200**. The user can program the ECC module **506** to function in accordance with any particular ECC algorithm selected by the user.

Tracking Data

File system **305** uses logical sector addressing to read and store information on flash memory medium **100/200**. Logical sector addresses are address locations that the file system reads and writes data to. They are "logical" because they are relative to the file system. In actuality, data may be stored in completely different physical locations on the flash memory medium **100/200**. These physical locations are referred to as physical sector addresses.

The flash driver **306** is responsible for linking all logical sector address requests (i.e., read & write) to physical sector address requests. The process of linking logical-to-physical sector addresses is also referred to herein as mapping. Going from logical to physical sector addresses permits flash driver **306** to have maximum flexibility when deciding where to store data on the flash memory medium **100/200**. The logical-to-physical sector mapping module **404** permits data to be flexibly assigned to any physical location on the flash memory medium, which provides efficiency for other tasks, such as wear-leveling and recovering from a power failure. It also permits the file system **305** to store data in the fashion it is designed to do so, without needing intelligence to know that the data is actually being stored on a flash medium in a different fashion.

FIG. **6A** shows an exemplary implementation of a data structure (i.e., a table) **600A** generated by the flash driver **306**. The data structure **600A** is stored in a volatile portion of memory **304**, e.g. RAM. The data structure **600A** includes physical sector addresses **602** that have a corresponding logical sector address **604**. An exemplary description of how table **600A** is generated is described with reference to FIG. **7**.

FIG. **7** illustrates a process **700** used to track data on the flash memory medium **100**/**200** when the file system **305** issues write requests to the flash driver **306**. Process **700** includes steps **702**–**718**. Referring to FIGS. **6**A and **7**, in step **702**, flash abstraction logic **308** receives a request to write data to a specified logical sector address **604**.

In step **704**, the sector manager **402** ascertains a free physical sector address location on the flash medium **100**/**200** that can accept data associated with the write request (how the sector manager **402** chooses physical sector addresses will be explained in more detail below). A free physical sector is any sector that can accept data without the need to be erased first. Once the sector manager **402** receives the physical sector address associated with a free physical sector location, the logical-to-physical sector mapping module **404** assigns the physical sector address to the logical sector address **604** specified by write request forming a corresponding relationship. For example, a physical sector address of **0** through N can be assigned to any arbitrary logical sector address **0** through N.

Next, in step **706**, the logical-to-physical sector mapping module **404** stores the corresponding relationship of the physical sector address to the logical sector address in a data structure, such as the exemplary table **600**A in memory **305**. As shown in the exemplary data structure **600**A, three logical sector addresses **604** are assigned to corresponding physical sector addresses **602**.

Next, in step **708** data associated with the logical sector address write request is stored on the flash medium **100**/**200** at the physical sector address location assigned in step **704**. For example, data would be stored in physical sector address location of zero on the medium **100**/**200**, which corresponds to the logical sector address of **11**.

Now, in step **710**, suppose for example purposes the file system **305** issues another write request, but in this case, to modify data associated with a logical sector address previously issued in step **702**. Then, flash driver **306** performs steps **712** through **714**, which are identical to steps **704** through **708**, respectively, which are described above.

In step **718**, however, after the updated data associated with step **710** is successfully stored on the flash medium **100**/**200**, the logical-to-physical sector mapping module **404** marks the old physical sector address assigned in step **704** as "dirty." Old data is marked dirty after new data is written to the medium **100**/**200**, so in the event there is a power failure in the middle of the write operation, the logical-to-physical sector mapping module **404** will not lose old data. It is possible to lose new or updated data from steps **702** or **710**, but since there is no need to perform an erase operation only one item of new or modified data is lost in the event of a power failure.

FIG. **6**B shows a data structure **600**B which is the same as data structure **600**A, except its contents have been updated. In this example the file system **305** has updated data associated with logical sector address **11**. Accordingly, the flash driver **306** reassigns logical sector address **11** to physical sector address **3** and stores the reassigned corresponding relationship between the these two addresses in data structure **600**B. As illustrated in data structure **600**B, the contents of logical sector **11** are actually written to physical sector address **3** and the contents of sector **0** are marked "dirty" after the data contents are successfully written into physical sector address **3** as was described with reference to steps **710**–**718**.

This process of reassigning logical-to-physical sector address when previously stored data is updated by the file system **305**, permits write operations to take place without having to wait to move an entire block of data and perform an erase operation. So, process **700** permits the data structure to be quickly updated and then the physical write operation can occur on the actual physical medium **100**/**200**. Flash abstraction logic **308** uses the data structures, such as **600**A/**600**B, to correctly maintain logical-to-physical mapping relationships.

When there is a read request issued by the files system **305**, the flash abstraction logic **308**, through the logical-to-physical mapping module **404**, searches the data structure **600**A/**600**B to obtain the physical sector address which has a corresponding relationship with the logical sector address associated with read request. The flash medium logic **310** then uses that physical sector address as a basis to send data associated with the read request back to the file system **305**. The file system **305** does not need intelligence to know that its requests to logical sector addresses are actually mapped to physical sector addresses.

Power-Interruption Protection

Write operations are performed at the sector-level as opposed to the block-level, which minimizes the potential for data loss during a power-failure situation. A sector worth of data is the finest level of granularity that is used with respect to most file systems **305**. Therefore, if the flash driver **306** is implemented to operate at a per sector basis, the potential for data loss during a power failure is reduced.

As mentioned above, data structures **600**A, **600**B are stored in memory **304**, which in one exemplary implementation is typically a volatile memory device subject to complete erasure in the event of a power failure. To safeguard data integrity on the flash medium **100**/**200**, logical-to-physical mapping information stored in the data structures **600**A/**600**B is backed-up on the flash memory medium.

In one exemplary implementation, to reduce the cost associated with storing the entire data structure on the flash memory medium **100**/**200**, the logical sector address is stored in the spare **104** area of the medium with each physical sector in which the logical sector address has a corresponding relationship.

FIG. **8** illustrates a process **800** for safeguarding mapping of logical-to-physical sector address information stored in volatile data structures, such as exemplary data structures **600**A and **600**B. Process **800** includes steps **802**–**814**. The order in which the process is described is not intended to be construed as a limitation. Furthermore, the process can be implemented in any suitable hardware, software, firmware, or combination thereof. In step **802**, the logical sector address associated with the actual data is stored in the physical sector of the flash memory medium **100**/**200** at the physical sector address assigned to the logical sector address. In the case of a NAND flash memory medium **100**, the logical sector address is stored in the spare area **104** of the medium. Using this scheme, the logical-to-physical sector mapping information is stored in a reverse lookup format. Thus, after a power failure situation, it is necessary to scan the spare area for each physical sector on the media, determine the corresponding logical sector address, and then update the in-memory lookup table accordingly. FIG. **9** illustrates a location with in media **100**/**200** in which the logical sector address can be stored. As previously mentioned, blocks of NOR flash memory can be logically subdivided into physical sectors each with a spare area (similar to NAND). Using this technique, the logical sector address is stored in the spare area for each the physical sector similar to the process used with NAND flash memory (shown in FIG. **15** as space **1504** to be described with reference to FIG. **15**).

In the event there is a power interruption and the data structures **600**A, **600**B are lost, as indicated by the YES branch of decisional step **804** of FIG. **8**, then flash abstraction logic **308** uses the flash medium logic **310** to scan the flash memory medium to locate the logical sector address stored with data in each physical address (see FIG. **9**), as indicated in step **806**. In step **808**, the physical sector address in which data is contained is reassigned to the logical sector address located with the data on the medium. As the physical and logical sector address are reestablished they are stored back in the data structures **600**A, **600**B and the flash medium logic **310** goes to the next sector containing data as indicated in step **812**. Steps **806**–**812** repeat until all sectors containing data have been are scanned and the data structure is reestablished. Normally, this occurs at initialization of the computer device **300**.

Accordingly, when a power failure occurs, process **800** enables the flash abstraction logic **308** to scan the medium **100**/**200** and rebuild the logical-to-physical mapping in a data structure such as the exemplary data structure **600**. Process **800** ensures that mapping information is not lost during a power failure and that integrity of the data is retained.

Dynamic Look-up Data Structure for Tracking Data

FIG. **10** illustrates a dynamic look-up data structure **1000** to track data stored in the flash memory medium **100**/**200**. Data structure **1000** includes a master data structure **1002** and one or more secondary data structures **1004**, **1006**. The data structures are generated and maintained by the flash driver **306**. The data structures are stored in a volatile portion of memory **304**. The one or more secondary tables **1004**, **1006** contain mappings of logical-to-physical sector addresses. Each of the secondary data structures **1004**, **1006**, as will be explained, have a predetermined capacity of mappings. The master data structure **1002** contains a pointer to each of the one or more secondary data structures **1004**, **1006**. Each secondary data structure is allocated on as needed basis for mapping those logical-to-physical addresses that are used to store data. Once the capacity of a secondary data structure **1004**, **1006**, etc., is exceeded, another secondary data structure is allocated, and another, etc., until eventually all possible physical sector addresses on the flash medium **100**/**200** are mapped to logical sector addresses. Each time a secondary table is allocated, a pointer contained in the master data structure **1002** is enabled by the flash driver **306** to point to it.

Accordingly, the flash driver **306** dynamically allocates one or more secondary data structures **1004**, **1006** based on the amount of permanent data stored on the flash medium itself. The size characteristics of the secondary data structures are computed at run-time using the specific attributes of the flash memory medium **100**/**200**. Secondary data structures are not allocated unless the secondary data structure previously allocated is full or insufficient to handle the amount of logical address space required by the file system **305**. Dynamic look-up data structure **1000**, therefore, minimizes usage of memory **304**. Dynamic look-up data structure **1000** lends itself to computer devices **300** that use calendars, inboxes, documents, etc. where most of the logical sector address space will not need to be mapped to a physical sector address. In these applications, only a finite range of logical sectors are repeatedly accessed and new logical sectors are only written when the application requires more storage area.

The master data structure **1002** contains an array of pointers, **0** through N that point to those secondary data structures that are allocated. In the example of FIG. **10**, the pointers at location **0** and **1** point to secondary data structures **1004** and **1006**, respectively. Also, in the example illustration of FIG. **10**, pointers **2** through N do not point to any secondary data structures and would contain a default setting, "NULL", such that the logical-to-physical sector mapping module **404** knows that there are no further secondary data structures allocated.

Each secondary data structure **1004**, **1006** is similar to data structures **600**, but only a portion of the total possible medium is mapped in the secondary data structures. The secondary data structures permit the flash abstraction logic **308** to reduce the amount space needed in memory **304**, to only those portions of logical sectors addresses issued by the file system. Each secondary data structure is (b*k) bytes in size, where k is the number of physical sector addresses contained in the data structure and b is the number of bytes used to store each physical sector address.

FIG. **11** illustrates a process **1100** for dynamically allocating look-up data structures for tracking data on the flash memory medium **100**/**200**. Process **1100** includes steps **1102** through **1106**. The order in which the process is described is not intended to be construed as a limitation. Furthermore, the process can be implemented in any suitable hardware, software, firmware, or combination thereof.

In step **1102**, a master data structure **1002** containing the pointers to one or more secondary data structures **1004**, **1006** is generated. The master data structure **1002** in this exemplary implementation is fixed in size. At the time the computer device **300** boots-up, the flash medium logic **310** determines the size of the flash memory medium **100**/**200** and relays this information to the flash abstraction logic **308**. Based on the size of the flash medium, the flash abstraction logic **308** calculates a range of physical addresses. That is, suppose the size of the flash medium is 16 MB, then a NAND flash medium **100** will typically contain 32768 sectors each 512 bytes in size. This means that the flash abstraction logic **308** may need to map a total of 0 through 32768 logical sectors in a worse case scenario, assuming all the memory space is used on the flash medium. Knowing that there are 2^{15} sectors on the medium, the flash abstraction logic **308** can use 2 bytes to store the physical sector address for each logical sector address. So the master data structure is implemented as an array of 256 DWORDs (N=256), which covers the maximum quantity of logical sector addresses (e.g., 32768) to be issued by the files system. So, there are a total of 256 potential secondary data structures.

In step **1104** the secondary data structure(s) are allocated. First, the flash abstraction logic determines the smallest possible size for each potential secondary data structure. Using simple division, 32768/256=128 logical sector addresses supported by each data structure. As mentioned above, the entire physical space can be mapped using 2 bytes, b=2, therefore, each secondary data structure will by 256 bytes in size or (b=2 * k=128).

Now, knowing the size of each secondary data structure, suppose that the file system **305** requests to write to logical sector addresses **50**–**79**, also known as LS**50**–LS**79**. To satisfy the write requests from the files system **305**, the flash abstraction logic **308** calculates that the first pointer in master data structure **1002** is used for logical sector addresses LS**0**–LS**127** or data structure **1004**. Assuming the first pointer is NULL, the flash abstraction logic **308** allocates data structure **1004** (which is 256 bytes in size) in memory **304**. As indicated in step **1106**, the flash abstraction logic **308** enables the pointer in position **0** of the master data structure to point to data structure **1004**. So, in this example,

data structure **1004** is used to store the mapping information for logical sectors LS**50**–LS**79**.

The flash abstraction logic **308** allocates a secondary data structure, if the file system **305** writes to the corresponding area in the flash medium **100**/**200**. Typically, only the logical sector addresses that are used are mapped by the flash abstraction logic **308**. So, in the worst case scenario, when the file system **305** accesses the entire logical address space, then all 256 secondary data structures (only two, **1004**, **1006** are shown to be allocated in the example of FIG. **10**), each 256 bytes in size will be allocated requiring a total of 64 KB of space in memory **304**.

When an allocated data structure **1004**, for instance, becomes insufficient to store the logical sector address space issued by the file system **305**, then the flash abstraction logic **308** allocates another data structure, like data structure **1006**. This process of dynamically allocating secondary data structures also applies if data structure **1004** becomes sufficient at a later time to again handle all the logical sector address requests made by the file system. In this example, the pointer to data structure **1006** would be disabled by the flash abstraction logic **308**; and data structure **1006** would become free space in memory **304**.

Uniform Wear Leveling and Recycling of Sectors

FIG. **12** is a diagram of flash memory medium **100**/**200** viewed and/or treated as a continuous circle **1200** by the flash driver **306**. Physically the flash memory media is the same as either media **100**/**200** shown in FIGS. **1** and **2**, except the flash abstraction logic **308**, organizes the flash memory medium as if it is a continuous circle **1200**, containing **0**-to-N blocks. Accordingly, the highest physical sector address (individual sectors are not shown in FIG. **12** to simplify the illustration, but may be seen in FIGS. **1** and **2**) within block N and the lowest physical sector address within block **0** are viewed as being contiguous.

FIG. **13** illustrates another view of media **100**/**200** viewed as a continuous circle **1200**. In this exemplary illustration, the sector manager **402** maintains a write pointer **1302**, which indicates a next available free sector to receive data on the medium. The next available free sector is a sector that can accept data without the need to be erased first in a prescribed order. The write pointer **1102** is implemented as a combination of two counters: a sector counter **1306** that counts sectors and a block counter **1304** that counts blocks. Both counters combined indicate the next available free sector to receive data.

In an alternative implementation, the write pointer **1302** can be implemented as a single counter and indicate the next physical sector that is free to accept data during a write operation. According to this implementation, the sector manager **402** maintains a list of all physical sector addresses free to receive data on the medium. The sector manager **402** stores the first and last physical sector addresses (the contiguous addresses) on the medium and subtracts the two addresses to determine an entire list of free sectors. The write pointer **1302** then advances through the list in a circular and continuous fashion. This reduces the amount of information needed to be stored by the sector manager **402**.

FIG. **14** illustrates a process **1400** used by the sector manager **402** to determine the next available free sector location for the flash driver **306** to store data on the medium **100**/**200**. Process **1400** also enables the sector manager **402** to provide each physical sector address (for the next free sector) for assignment to each logical sector address write request by the file system **305** as described above. Process **1400** includes steps **1402**–**1418**. The order in which the process is described is not intended to be construed as a limitation. Furthermore, the process can be implemented in any suitable hardware, software, firmware, or combination thereof.

In step **1402**, the X block counter **1304** and Y sector counter **1306** are initially set to zero. At this point it is assumed that no data resides on the medium **100**/**200**.

In step **1404**, the driver **306** receives a write request and the sector manager **402** is queried to send the next available free physical sector address to the logical-to-physical sector mapping module **404**. The write request may come from the file system **305** and/or internally from the compactor **406** for recycling sectors as shall be explained in more detail below.

In step **1406**, the data is written to the sector indicated by the write pointer **1302**. Since both counters are initially set to zero in this exemplary illustration, suppose that the write pointer **1302** points to sector zero, block zero.

In step **1408**, the sector counter **1306** is advanced one valid sector. For example, the write pointer advances to sector one of block zero, following the example from step **1406**.

Next, in decisional step **1410**, the sector manager **402** checks whether the sector counter **1306** exceeds the number of sectors K in a block. If the Y count does not exceed the maximum sector size of the block, then according to the NO branch of decisional step **1410**, steps **1404**–**1410** repeat for the next write request.

On the other hand, if the Y count does exceed the maximum sector size of the block, then the highest physical sector address of the block was written to and the block is full. Then according to the YES branch of step **1410**, in step **1412** the Y counter is reset to zero. Next, in step **1414**, X block counter **1304** is incremented by one, which advances the write pointer **1302** to the next block at the lowest valid physical sector address, zero, of that block.

Next, in decisional step **1416**, the compactor **406** checks whether the X block counter is pointing to a bad block. If it is, X block counter **1304** is incremented by one. In one implementation, the compactor **406** is responsible for checking this condition. As mentioned above, the sector manager stores all of the physical sector addresses that are free to handle a write request. Entire blocks of physical sector addresses are always added by the compactor during a compaction or during initialization. So, the sector manager **402** does not have to check to see if blocks are bad, although the sector manager could be implemented to do so. It should also be noted that in other implementations step **1416** could be performed at the start of process **1400**.

In step **1417**, the X block counter **1304** is incremented until it is pointing to a good block. To avoid a continuous loop, if all the blocks are bad, then process **1400** stops at step **1416** and provides an indication to a user that all blocks are bad.

Next in decisional step **1418**, the sector manager checks whether the X block counter **1304** exceeds the maximum numbers of blocks N. This would indicate that write pointer **1302** has arrived full circle (at the top of circle **1200**). If that is the case, then according to the YES branch of step **1418**, the process **1400** repeats and the X and Y counter are reset to zero. Otherwise, according to the NO branch of step **1418**, the process **1400** returns to step **1404** and proceeds.

In this exemplary process **1400**, the write pointer **1302** initially starts with the lowest physical sector address of the lowest addressed block. The write pointer **1302** advances a sector at a time through to the highest physical sector address of the highest addressed block and then back to the lowest, and so forth. This continuous and circular process **1400** ensures that data is written to each sector of the

medium **100/200** fairly and evenly. No particular block or sector is written to more than any other, ensuring even wear-levels throughout the medium **100/200**. Accordingly, process **1400** permits data to be written to the next available free sector extremely quickly without expensive processing algorithms used to determine where to write new data while maintaining even wear-levels. Such conventional algorithms can slow the write speed of a computer device.

In an alternative implementation, it is possible for the write pointer **1302** to move in a counter clock wise direction starting with highest physical sector address of the highest block address N and decrement its counters. In either case, bad blocks can be entirely skipped and ignored by the sector manager. Additionally, the counters can be set to any value and do not necessarily have to start with the highest or lowest values of for the counters.

FIG. **15** illustrates another view of media **100/200** viewed as a continuous circle **1200**. As shown in FIG. **15**, the write pointer **1302** has advanced through blocks **0** through **7** and is approximately half way through circle **1200**. Accordingly, blocks **0** through **7** contain dirty, valid data, or bad blocks. That is, each good sector in blocks **0** through **7** is not free, and therefore, not available to receive new or modified data. Arrow **1504** represents that blocks **0** through **7** contain used sectors. Eventually, the write pointer **1302** will either run out of free sectors to write to unless sectors that are marked dirty or are not valid are cleared and recycled. To clear a sector means that sectors are reset to a writable state or in other words are "erased." In order to free sectors it is necessary to erase at least a block at a time. Before a block can be erased, however, the contents of all good sectors are copied to the free sectors to a different portion of the media. The sectors are then later marked "dirty" and the block is erased.

The compactor **406** is responsible for monitoring the condition of the medium **100/200** to determine when it is appropriate to erase blocks in order to recycle free sectors back to the sector manager **402**. The compactor **406** is also responsible for carrying out the clear operation. To complete the clear operation, the compactor **406**, like the sector manager **402**, maintains a pointer. In this case, the compactor **406** maintains a clear pointer **1502**, which is shown in FIG. **15**. The clear pointer **1502** points to physical blocks and as will be explained enables the compactor **406** to keep track of sectors as the medium **100/200** as blocks are cleared. The compactor **406** can maintain a pointer to a block to compact next since an erase operation affects entire blocks. That is, when the compactor **406** is not compacting a block, the compactor **406** points to a block.

FIG. **16** is a flow chart illustrating a process **1600** used by the compactor to recycle sectors. Process **1600** includes steps **1602–1612**. The order in which the process is described is not intended to be construed as a limitation. Furthermore, the process can be implemented in any suitable hardware, software, firmware, or combination thereof. In step **1602**, the compactor **406** monitors how frequently the flash memory medium **100/200** is written to or updated by the file system. This is accomplished by specifically monitoring the quantities of free and dirty sectors on the medium **100/200**. The number of free sectors and dirty sectors can be determined counting free and dirty sectors stored in tables **600** and/or **900** described above.

In decisional step **1604**, the compactor **406** performs two comparisons to determine whether it is prudent to recycle sectors. The first comparison involves comparing the amount of free sectors to dirty sectors. If the amount of dirty sectors outnumbers the free sectors, then the compactor **406** deems it warranted to perform a recycling operation, which in this case is referred to as a "service compaction." Thus a service compaction is indicated when the number of dirty sectors outnumbers the quantity of free sectors.

If a service compaction is deemed warranted, then in step **1606** the compactor waits for a low priority thread **1606**, before seizing control of the medium to carry out steps **1608–1612** to clear blocks of dirty data. The service compaction could also be implemented to occur at other convenient times when it is optional to recycle dirty sectors into free sectors. For instance, in an alternative implementation, when one third of the total sectors are dirty, the flash abstraction logic **308** can perform a service compaction. In either implementation, usually the compactor **406** waits for higher priority threads to relinquish control of the processor **302** and/or flash medium **100/200**. Once a low priority thread is available, the process proceeds to step **1608**.

Referring back to step **1604**, the second comparison involves comparing the amount of free sectors left on the medium, to determine if the write pointer **1302** is about to or has run out of free sectors to point to. If this is the situation, then the compactor **406** deems it warranted to order a "critical compaction" to recycle sectors. The compactor does not wait for a low priority thread and launches immediately into step **1608**.

In step **1608**, the compactor **406** operates at either a high priority thread or low priority thread depending on step **1604**. If operating at a high level thread (critical compaction), the compactor **1102** is limited to recycling a small number, e.g., 16 dirty sectors, into free sectors and return control of the processor back to computer device **300** to avoid monopolizing the processor **302** during such an interruption.

Thirty two sectors per block is commonly manufactured for flash media, but other numbers of sectors, larger or smaller, could be selected for a critical compaction. Regardless of these size characteristics, the number of sectors recycled during a critical compaction is arbitrary but must be at least 1 (in order to satisfy the current WRITE request). A critical compaction stalls the file system **305** from being able to complete a write; therefore, it is important to complete the compaction as soon as possible. In the case of a critical compaction, the compactor **406** must recycle at least one dirty sector into a free sector so that there is space on the medium to fulfill the pending write request. Having more than one sector recycled at a time, such as 16, avoids the situation where there are multiple pending write requests and multiple critical compactions that are performed back-to-back, effectively blocking control of the processor indefinitely. So, while the number of sectors recycled chosen for a critical compaction can vary, a number sufficient to prevent back-to-back critical compactions is implemented in the exemplary description.

So, in step **1608**, the compactor **406** will use the clear pointer **1502** to scan sectors for valid data, rewrite the data to free sectors, and mark a sector dirty after successfully moving data. Accordingly, when moving data, the compactor uses the same processes described with reference to process **700**, which is the same code that is used when the file system **305** writes new and/or updates data. The compactor **406** queries the sector manager **402** for free sectors when moving data, in the same fashion as described with reference to process **1400**.

In step **1610**, the compactor **406** moves the clear pointer **1502** sector-by-sector using a sector counter like the write counter **1306** shown in FIG. **13**, except this sector counter pertains to the location of the clear pointer **1502**. The compactor **406** also keeps track of blocks through a counter

in similar fashion as described with reference to the write pointer **1302**. However, the amount of blocks cleared is determined by the number of dirty sectors with the exception of a critical compaction. In a critical compaction, the compactor only compacts enough blocks to recycle a small number of physical sectors (i.e. 16 sectors).

In step **1612**, the compactor erases (clears) those blocks which contain good sectors that are fully marked dirty. FIG. **17** shows exemplary results from process **1600**. In this example, blocks **0** and **1** were cleared and the clear pointer was moved to the first sector of block **2**, in the event another compaction is deemed warranted. As a result, the compactor **406** recycled two blocks worth of the sectors from blocks **0** and **1**, which provides more free sectors to the sector manager **402**. Used sectors **1504** forms a data stream (hereinafter a "data stream" **1504**) that rotates in this implementation in a clockwise fashion. The write pointer **1302** remains at the head of the data stream **1504** and the clear pointer **1502** remains at the end or "tail" of the data stream **1504**. The data stream **1504** may shrink as data is deleted, or grow as new data is added, but the pointers always point to opposite ends of the data stream **1504**: head and tail.

Treating the flash memory medium as if the physical sector addresses form a continuous circle **1200**, and using the processes described above, enables the flash abstraction logic **308** to accomplish uniform wear-leveling throughout the medium **100/200**. The compactor **406** selects a given block the same number times for recycling of sectors through erasure. Since flash blocks have a limited write/erase cycle, the compactor as well as the sector manager distributes these operations across blocks **0**–N as evenly and as fairly as possible. In this regard, the data steam **1504** rotates in the circle **1200** (i.e. the medium **100/200**) evenly providing perfect wear-levels on the flash memory medium **100/200**.

In the event of power failure, the flash abstraction logic **310** contains simple coded logic that scans the flash memory medium **100/200** and determines what locations are marked free and dirty. The logic is then able to deduce that the data stream **1504** resides between the locations marked free and dirty, e.g. . . . , the data stream **1106** portion of the circle **1200** described in FIG. **17**. The head and tail of the data stream **1504** is easily determined by locating the highest of the physical sector addresses containing data for the head and by locating the lowest of the physical sector addresses containing data for the tail.

NOR Flash Devices

Although all the aforementioned sections in this Detailed Description section apply to NAND and NOR flash devices, if a NOR flash memory medium **200** is used, some additional implementation is needed for the flash medium logic to support the storing of data in each physical sector on the medium **200**. Each NOR block **0**, **1**, **2**, etc. can be treated like a NAND flash memory medium **100**, by the flash medium logic **310**. Specifically, each NOR block is subdivided into some number of pages where each page consists of a 512 byte "data area" for sector data and an 8 byte "spare area" for storing things like to the logical sector address, status bits, etc. (as described above).

FIG. **18** illustrates a logical representation of a NOR flash memory medium **200** divided in way to better support the processes and techniques implemented by the flash driver. In this implementation, sectors **1802** contain a 512 byte data area **1803** for the storage of sector related data and 8 bytes for a spare area **1804**. Sections **1806** represent unused portions of NOR blocks, because a NOR Flash block is usually a power of 2 in size, which is not evenly divisible. For instance, consider a 16 MB NOR flash memory device that has 128 flash blocks each 128 KB in size. Using a page size equal to 520 bytes, each NOR flash block can be divided into 252 distinct sectors with 32 bytes remaining unused. Unfortunately, these 32 bytes per block are "wasted" by the flash medium logic **310** in the exemplary implementation and are not used to store sector data. The tradeoff, however, is the enhanced write throughput, uniform wear leveling, data loss minimization, etc. all provided by the flash abstraction logic **308** of the exemplary flash driver **306** as described above. Alternative implementations could be accomplished by dividing the medium **200** into different sector sizes.

Computer Readable Media

An implementation of exemplary subject matter using a flash driver as described above may be stored on or transmitted across some form of computer-readable media. Computer-readable media can be any available media that can be accessed by a computer. By way of example, and not limitation, computer readable media may comprise "computer storage media" and "communications media."

"Computer storage media" include volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules, or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by a computer.

"Communication media" typically embodies computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as carrier wave or other transport mechanism. Communication media also includes any information delivery media.

The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared, and other wireless media. Combinations of any of the above are also included within the scope of computer readable media.

Conclusion

Although the invention has been described in language specific to structural features and/or methodological acts, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the claimed invention.

What is claimed is:

1. A method for managing erasures on a flash memory device having addressable physical sectors organized as blocks, comprising:

tracking how many physical sectors are free to receive data;

tracking how many physical sectors contain data that is dirty;

comparing whether the physical sectors that are free to receive data outnumber the physical sectors that contain data that is dirty; and

executing an erase operation of one or more blocks when the physical sectors that contain data that is dirty outnumber the physical sectors that are free to receive data.

2. The method as recited in claim **1**, further comprising: scheduling the execution of the erase operation as a low priority thread.

3. The method as recited in claim **1**, further comprising:

tracking when the physical sectors that are free to receive data are insufficient in quantity to receive write requests from a file system; and

executing a second erase operation of one or more blocks if the physical sectors that are free to receive data are insufficient in quantity.

4. The method as recited in claim **3**, further comprising scheduling the execution of the second erase operation as high priority thread.

5. One or more computer-readable media comprising computer executable instructions that, when executed, perform the method as recited in claim **1**.

6. A method for managing erasures on a flash memory device having addressable physical sectors organized as blocks, comprising:

tracking how many physical sectors are free to receive data;

tracking when the physical sectors that are free to receive data are insufficient in quantity to receive write requests from a file system; and

executing an erase operation of one or more blocks if the physical sectors that are free to receive data are insufficient in quantity.

7. The method as recited in claim **6**, further comprising scheduling the execution of the erase operation as high priority thread.

8. The method as recited in claim **6**, further comprising:

tracking how many physical sectors contain data that is dirty;

comparing whether the physical sectors that are free to receive data outnumber the physical sectors that contain data that is dirty; and

executing a second erase operation of one or more blocks when the physical sectors that contain data that is dirty outnumber the physical sectors that are free to receive data.

9. The method as recited in claim **8**, further comprising: scheduling the execution of the second erase operation as a low priority thread.

10. One or more computer-readable media comprising computer executable instructions that, when executed, perform the method as recited in claim **6**.

11. A system for flash memory having addressable locations organized as blocks, comprising:

flash abstraction logic, configured to track how many physical sectors are free to receive data; track how many physical sectors contain data that is dirty, and compare whether the physical sectors that are free to receive data outnumber the physical sectors that contain data that is dirty; and

a compactor, configured to perform an erase operation of one or more blocks when the physical sectors that contain data that is dirty outnumber the physical sectors that are free to receive data.

12. The system as recited in claim **11**, wherein when the compactor performs an erase operation, the compactor advances through a sequence of the physical sector addresses and if a particular physical sector address corresponding to a physical sector in the memory contains valid data, the compactor moves the valid data to a physical sector address corresponding to a physical sector in the memory that is free to receive data before performing the erase operation.

13. The system as recited in claim **11**, wherein the compactor performs the erase operation as low priority thread.

14. The system as recited in claim **11**, wherein the flash abstraction logic is further configured to track how many physical sector addresses are free to receive data,

track when the physical sector addresses that are free to receive data are insufficient in quantity to receive write requests from a file system; and

the compactor is further configured to execute an erase operation of one or more blocks if the physical sector addresses that are free to receive data are insufficient in quantity.

15. A system for flash memory having addressable locations organized as blocks, comprising:

flash abstraction logic configured to track how many physical sector addresses are free to receive data, and track when the physical sector addresses that are free to receive data are insufficient in quantity to receive write requests from a file system; and

a compactor, configured to execute an erase operation of one or more blocks if the physical sector addresses that are free to receive data are insufficient in quantity.

16. The system as recited in claim **15**, wherein when the compactor performs an erase operation, the compactor advances through a sequence of the physical sector addresses and if a particular physical sector address corresponding to a physical sector in the memory contains valid data, the compactor moves the valid data to a physical sector address corresponding to a physical sector in the memory that is free to receive data before performing the erase operation.

17. The system as recited in claim **15**, wherein the flash abstraction logic is further configured to track how many physical sectors are free to receive data; track how many physical sectors contain data that is dirty, and compare whether the physical sectors that are free to receive data outnumber the physical sectors that contain data that is dirty; and

the compactor, is further configured to perform an erase operation of one or more blocks when the physical sectors that contain data that is dirty outnumber the physical sectors that are free to receive data.

18. The system as recited in claim **15**, wherein the compactor performs the erase operation as a high priority thread.

19. One or more computer-readable media comprising computer-executable instructions that, when executed by a computer, causes the computer to:

organize a flash memory having addressable locations into blocks;

track how many physical sectors are free to receive data;

track how many physical sectors contain data that is dirty;

compare whether the physical sectors that are free to receive data outnumber the physical sectors that contain data that is dirty; and

perform an erase operation of one or more blocks when the physical sectors that contain data that is dirty outnumber the physical sectors that are free to receive data.

20. One or more computer-readable media as recited in claim 19, that causes the computer when performing the erase operation to:

advance through a sequence of the physical sector addresses; and

if a particular physical sector address corresponding to a physical sector in the memory contains valid data, to move the valid data to a physical sector address corresponding to a physical sector in the memory that is free to receive data before performing the erase operation.

21. One or more computer-readable media as recited in claim 19, wherein the computer treats the erase operation as low priority thread.

22. One or more computer-readable media as recited in claim 19, that causes the computer to: further track how many physical sector addresses are free to receive data, track when the physical sector addresses that are free to receive data are insufficient in quantity to receive write requests from a file system; and perform an erase operation of one or more blocks if the physical sector addresses that are free to receive data are insufficient in quantity.

23. One or more computer-readable media comprising computer-executable instructions that, when executed by a computer, causes the computer to:

organize a flash memory having addressable locations into blocks;

track how many physical sector addresses are free to receive data;

track when the physical sector addresses that are free to receive data are insufficient in quantity to receive write requests from a file system; and

execute an erase operation of one or more blocks if the physical sector addresses that are free to receive data are insufficient in quantity.

24. One or more computer-readable media as recited in claim 23, that causes the computer when performing the erase operation to:

advance through a sequence of the physical sector addresses; and

if a particular physical sector address corresponding to a physical sector in the memory contains valid data, to move the valid data to a physical sector address corresponding to a physical sector in the memory that is free to receive data before performing the erase operation.

25. One or more computer-readable media as recited in claim 23, that causes the computer to further track how many physical sectors are free to receive data; track how many physical sectors contain data that is dirty, and compare whether the physical sectors that are free to receive data outnumber the physical sectors that contain data that is dirty; and perform an erase operation of one or more blocks when the physical sectors that contain data that is dirty outnumber the physical sectors that are free to receive data.

26. One or more computer-readable media as recited in claim 23, wherein the computer treats the erase operation as a high priority thread.

* * * * *

US006826762B2

(12) **United States Patent**
Shell et al.

(10) **Patent No.: US 6,826,762 B2**
(45) **Date of Patent: Nov. 30, 2004**

(54) **RADIO INTERFACE LAYER IN A CELL PHONE WITH A SET OF APIS HAVING A HARDWARE-INDEPENDENT PROXY LAYER AND A HARDWARE-SPECIFIC DRIVER LAYER**

(75) Inventors: **Scott R. Shell**, Redmond, WA (US); **Roman Sherman**, Bellevue, WA (US); **Alan W. Shen**, Seattle, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 602 days.

(21) Appl. No.: **09/788,317**

(22) Filed: **Feb. 16, 2001**

(65) **Prior Publication Data**

US 2002/0184407 A1 Dec. 5, 2002

(51) **Int. Cl.**[7] **G06F 1/12**

(52) **U.S. Cl.** **719/328**; 719/311; 719/313; 719/321; 719/322; 719/327; 719/329

(58) **Field of Search** 719/311, 313, 719/321, 322, 327, 328, 329; 709/310

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,018,571	A	1/2000	Langlois et al.	379/207
6,141,564	A	10/2000	Bruner et al.	455/558
6,269,254	B1 *	7/2001	Mathis	455/557
6,584,185	B1 *	6/2003	Nixon	379/201.01
2002/0052968	A1 *	5/2002	Bonefas et al.	709/231

FOREIGN PATENT DOCUMENTS

EP	0994614 A2	4/2000	H04L/29/06
WO	WO96/06393	2/1996	G06F/9/44

OTHER PUBLICATIONS

Tso, Mike et al., "Always On, Always Connected Mobile Computing," *Universal Personal Communications*, 1996 IEEE International Conference on Cambridge, MA, USA, Sep. 29–Oct. 2, 1996, New York, NY, pp. 918924.

Steeman, H., "Wireless Application Protocol (WAP)," *Elektor Electronics*, vol. 26, No. 289, Jun. 2000, pp. 5658.

Bridging Wireless and Wired Networks: Smart Phone Operating Systems, IP Convergence and Market Segmentation G. E. Darby; *Info vol. 1, No. 6*, 1999; pp. 563–576.

Design: Designing Mobile Phones and Communicators for Consumer Needs at Nokia; Kaisa VaananerVainio–Mattila and Satu Ruuska; *Interactions* 6,5 (Sep. 1999), pp. 23–26.

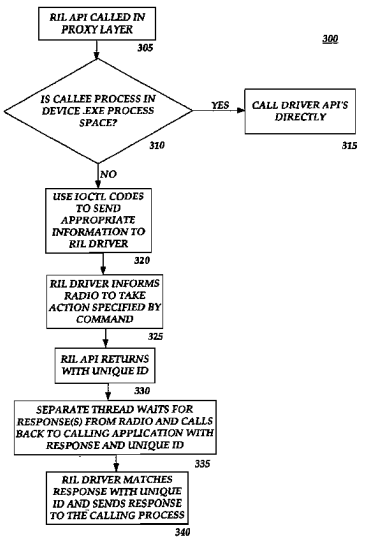
(List continued on next page.)

Primary Examiner—Nabil El-Hady
(74) *Attorney, Agent, or Firm*—Merchant & Gould

(57) **ABSTRACT**

A Radio Interface Layer (RIL) is disclosed. The RIL comprises an API set which provides a level of abstraction between the radio on a cell phone and the software of the cell phone. The API set of RIL is roughly based on the GSM AT interface as defined in GSM specifications 07.05 and 07.07. The API set provides access to functionality contained within a cellular telephone, such as a GSM or CDMA compatible telephone. These APIs allow applications running on an operating system in the cellular telephone to issue commands without knowledge of the underlying radio structure of the cellular telephone and specific knowledge of the GSM-type commands. For example, these APIs allow the applications to access to phonebook entries, restrict access to data and functionality using passwords, access file and message storage, and perform many other functions. The RIL is divided into a hardware-independent proxy layer, called by various software components, and a driver layer that is hardware-specific.

30 Claims, 4 Drawing Sheets



OTHER PUBLICATIONS

Making Place to Make IT Work: Empirical Explorations of HCI for Mobile CSCW, Steinar Kristoffersen and Frederik Ljungberg; *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work*, 1999, pp. 27–85.

Mobile Computing: Beyond Laptops; Laura Cappelletti; *Proceedings of the 15th Annual International Conference on Computer Documentation*; 1997; pp. 23–26.

L^2imbo: A Distributed Systems Platform for Mobile Computing; Nigel Davies, Adrian Friday, Stephen P. Wade and Gordon S. Blair; *Mob. Netw. Appl.* 3,2 (Aug. 1998), pp. 143–156.

* cited by examiner

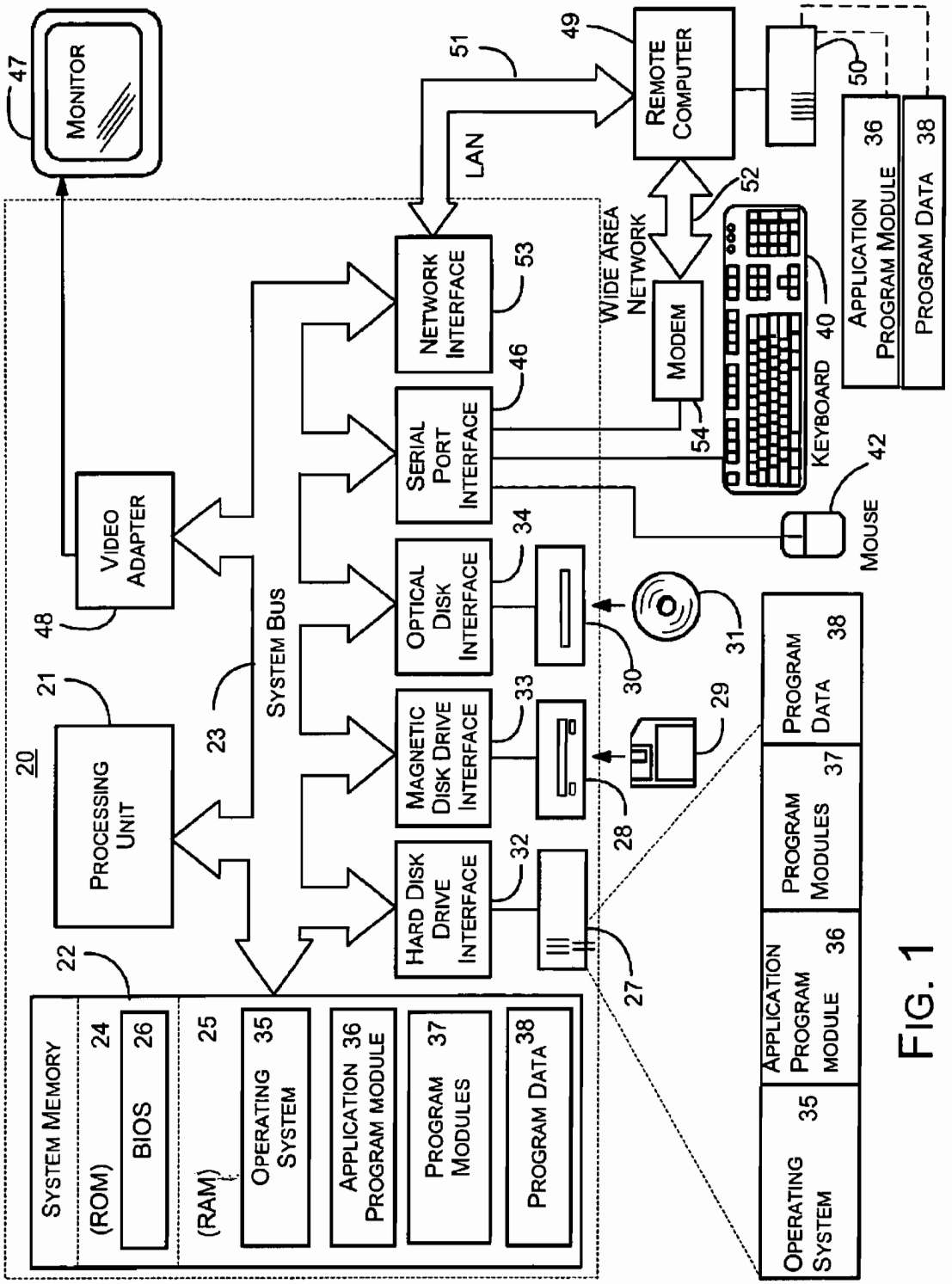

MONITOR
47
REMOTE COMPUTER
49
51
LAN
50
APPLICATION PROGRAM MODULE
36
PROGRAM DATA
38
WIDE AREA NETWORK
52
MODEM
54
KEYBOARD
40
MOUSE
42
NETWORK INTERFACE
53
SERIAL PORT INTERFACE
46
VIDEO ADAPTER
48
OPTICAL DISK INTERFACE
34
MAGNETIC DISK DRIVE INTERFACE
33
HARD DISK DRIVE INTERFACE
32
SYSTEM BUS
23
20
PROCESSING UNIT
21
22
SYSTEM MEMORY
(ROM)
24
BIOS
26
(RAM)
25
OPERATING SYSTEM
35
APPLICATION PROGRAM MODULE
36
PROGRAM MODULES
37
PROGRAM DATA
38
30
31
29
28
27
OPERATING SYSTEM
35
APPLICATION PROGRAM MODULE
36
PROGRAM MODULES
37
PROGRAM DATA
38

FIG. 1

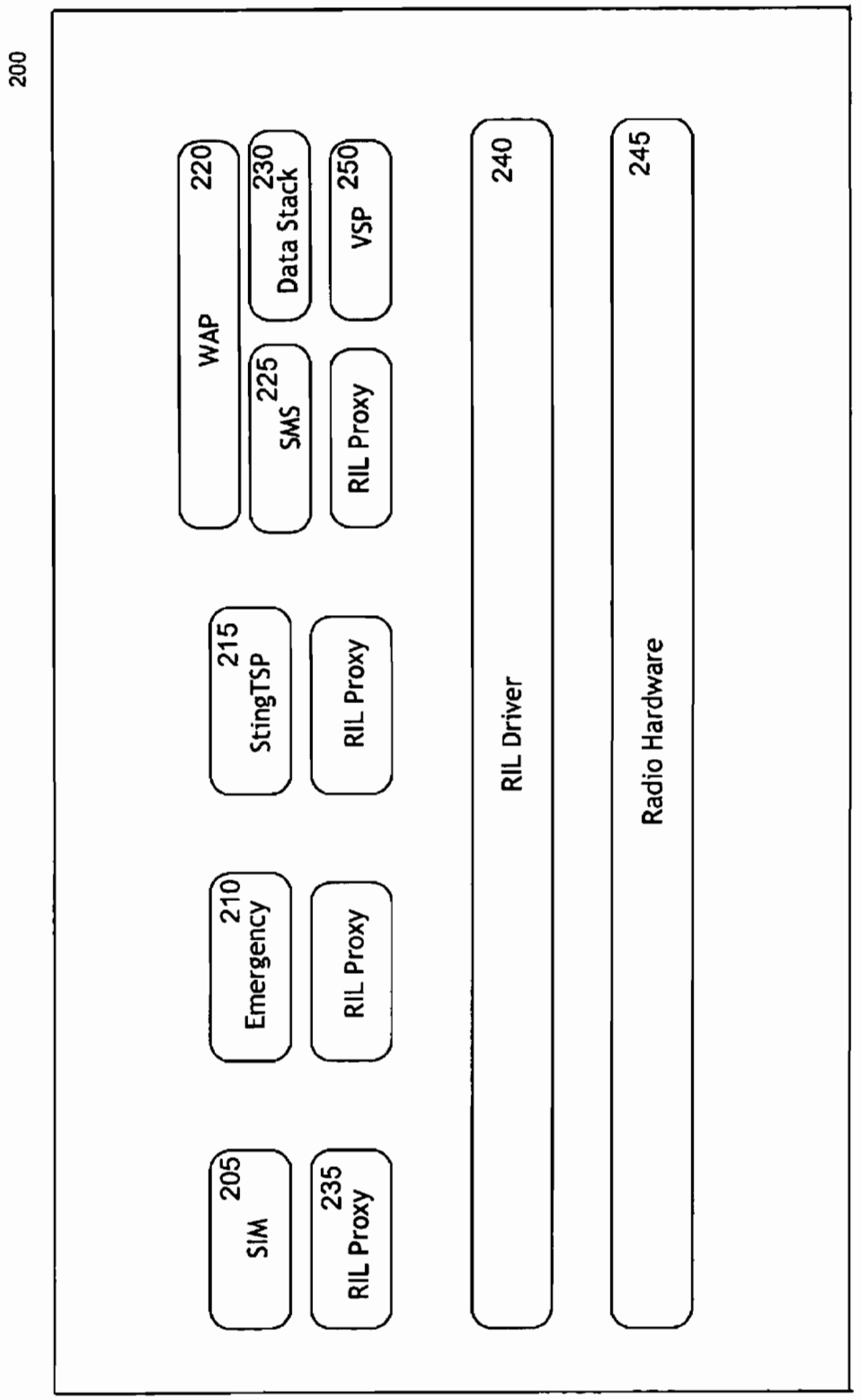

200
205
SIM
235
RIL Proxy
210
Emergency
RIL Proxy
215
StingTSP
RIL Proxy
240
RIL Driver
245
Radio Hardware
220
WAP
225
SMS
230
Data Stack
RIL Proxy
250
VSP

FIG. 2

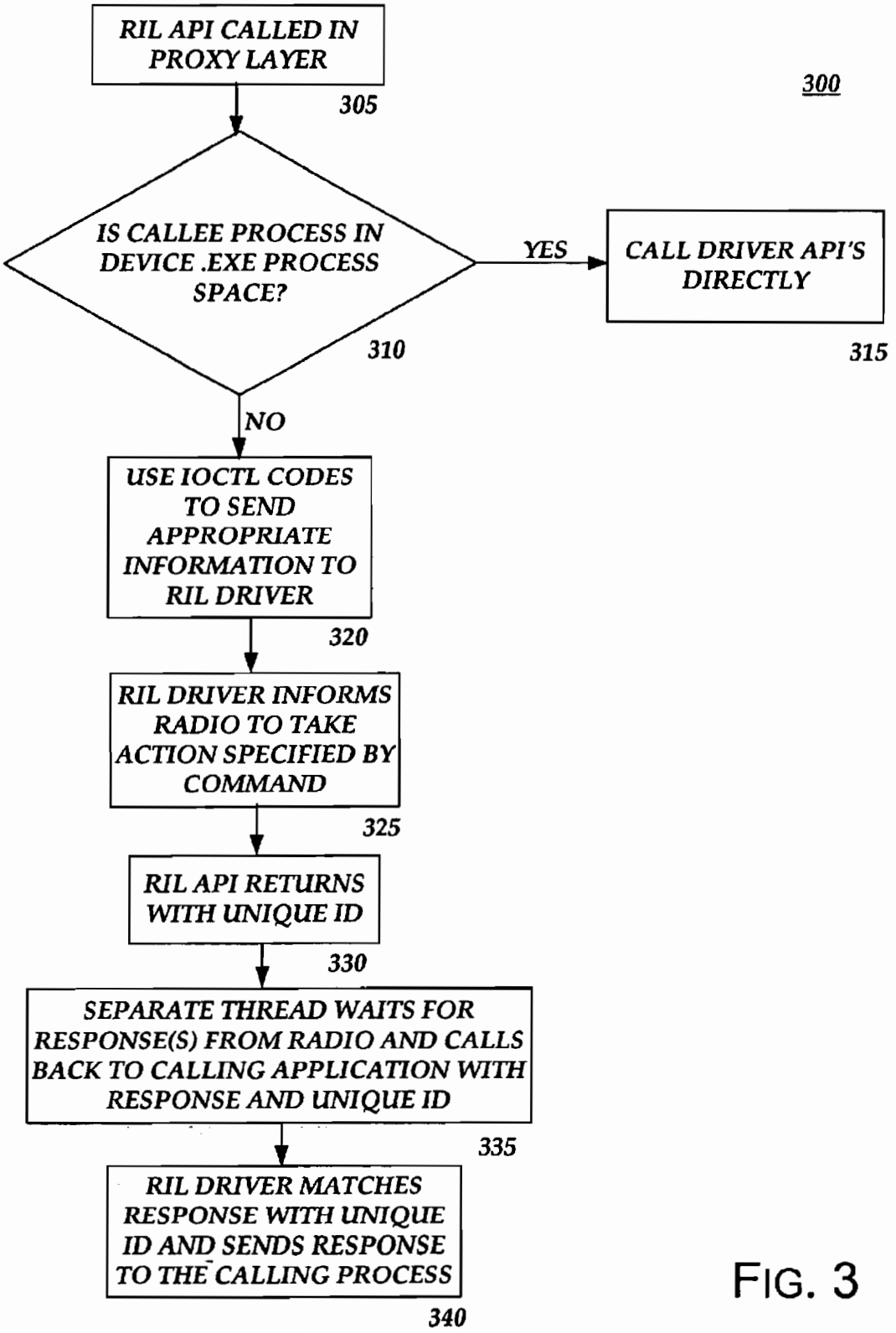

RIL API CALLED IN PROXY LAYER
305
300
IS CALLEE PROCESS IN DEVICE .EXE PROCESS SPACE?
310
YES
CALL DRIVER API'S DIRECTLY
315
NO
USE IOCTL CODES TO SEND APPROPRIATE INFORMATION TO RIL DRIVER
320
RIL DRIVER INFORMS RADIO TO TAKE ACTION SPECIFIED BY COMMAND
325
RIL API RETURNS WITH UNIQUE ID
330
SEPARATE THREAD WAITS FOR RESPONSE(S) FROM RADIO AND CALLS BACK TO CALLING APPLICATION WITH RESPONSE AND UNIQUE ID
335
RIL DRIVER MATCHES RESPONSE WITH UNIQUE ID AND SENDS RESPONSE TO THE CALLING PROCESS
340

FIG. 3

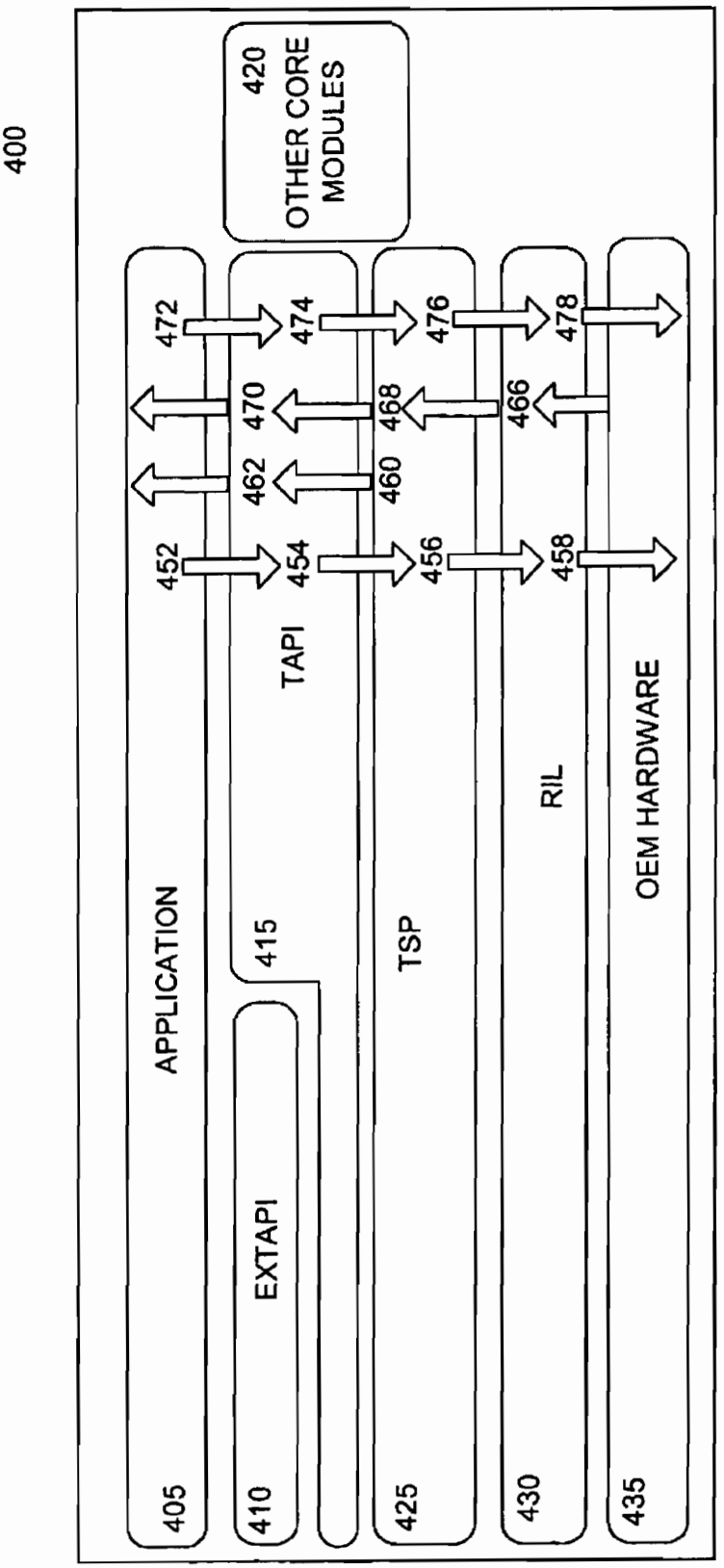

400
420
OTHER CORE MODULES
472
474
476
478
470
468
466
462
460
452
454
456
458
APPLICATION
TAPI
TSP
RIL
OEM HARDWARE
EXTAPI
415
405
410
425
430
435

FIG. 4

RADIO INTERFACE LAYER IN A CELL PHONE WITH A SET OF APIS HAVING A HARDWARE-INDEPENDENT PROXY LAYER AND A HARDWARE-SPECIFIC DRIVER LAYER

TECHNICAL FIELD

The invention generally relates to application programming interfaces (APIs) and, even more particularly, relates to a Radio Interface Layer comprising a set of APIs.

BACKGROUND

Cellular telephones are becoming commonplace in today's world. As users become more accustomed to using cellular telephones, they are requesting more sophisticated uses of telephones. Ideally, users would like their cellular telephones to perform the same functions as their personal computers or hand-held PDAs. Implementing such uses in a cellular telephone environment requires application developers to develop or adapt their software for use on a cellular telephone. However, adapting or developing software for use on one OEM's cellular telephone does not necessarily guarantee that the software application will function on another OEM's cellular telephone due to the different radio implementations of different OEMs and due to the differences in different cellular environments.

In order to create a software solution adaptable to multiple different cellular systems and radios, there is a need for some kind of a hardware adaptation layer, i.e. a layer that isolates the specifics of a particular cellular system/hardware from the bulk of the software system. There is a further need to expose a predefined interface used by the software components. There is still a further need that the layer should allow hardware manufacturers to replace/modify the implementation of the hardware interface to conform to their specific hardware.

Such a layer (TAPI) already exists for use in development of general telephony systems. However, TAPI has two disadvantages making it difficult to use in a cellular environment: a significant amount of cellular-specific functionality isn't exposed by the TAPI interface and TAPI Service Providers (TSPs) are quite difficult to implement, thus making it harder to adapt the software system to different types of hardware. Hence, there is a need for a new hardware adaptation layer which is more specifically suited to the cellular environment and which simplifies the task of adapting it to different types of hardware.

SUMMARY OF THE INVENTION

The present invention meets the above-described needs by providing a Radio Interface Layer (RIL), which is an API set providing a level of abstraction between the radio on a cell phone and the software of the cell phone. The API set of RIL is roughly based on the GSM AT interface as defined in GSM specifications 07.05 and 07.07. The API set provides access to functionality contained within a cellular telephone, such as a GSM or CDMA compatible telephone. The present invention allows applications running on an operating system in the cellular telephone to issue commands without knowledge of the underlying radio structure of the cellular telephone and without specific knowledge of the GSM-type commands. For example, the present invention allows the applications to access phonebook entries, restrict access to data and functionality using passwords, access file and message storage, and perform many other functions. The RIL is divided into a hardware-independent proxy layer, called by various software components, and a driver layer that is hardware-specific. It should be understood that an OEM may replace the driver layer with their own layer containing implementation specific to their hardware.

That the invention improves over the drawbacks of the prior art and accomplishes the advantages described above will become apparent from the following detailed description of the exemplary embodiments and the appended drawings and claims.

BRIEF DESCRIPTION OF THE FIGURES

FIG. **1** is a block diagram of an exemplary personal computer system.

FIG. **2** is a block diagram illustrating an exemplary embodiment of an RIL in a phone in accordance with an embodiment of the present invention.

FIG. **3** is a flow diagram illustrating a method for processing of commands using the radio interface layer (RIL) in accordance with an embodiment of the present invention.

FIG. **4** is a block diagram illustrating a method for an application to establish a voice call using RIL in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

An embodiment of the present invention will be incorporated into a cellular telephone marketed by Microsoft Corporation of Redmond, Wash. The cellular telephone may be a "smart phone" that, in addition to providing telephony services, also runs different software applications and performs different functions normally reserved for personal computers or PDAs. For example, in one embodiment, the telephone may be used as a personal information manager (PIM) for storing appointments, contacts, tasks, etc.

Other embodiments of the present invention may be incorporated into PDAs, personal computers and hand-held computers. FIG. **1** and the following discussion are intended to provide a brief, general description of an exemplary personal computer system for use with the above-described embodiments of the present invention. Those skilled in the art will recognize that software products may include routines, programs, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, software products may be located in both local and remote memory storage devices.

With reference to FIG. **1**, an exemplary system for implementing the invention includes a conventional personal computer **20**, including a processing unit **21**, a system memory **22**, and a system bus **23** that couples the system memory to the processing unit **21**. The system memory **22** includes read only memory (ROM) **24** and random access memory (RAM) **25**. A basic input/output system **26** (BIOS), containing the basic routines that help to transfer information between elements within the personal computer **20**,

such as during start-up, is stored in ROM **24**. A video BIOS **60** may also be stored in ROM **24**. The personal computer **20** further includes a hard disk drive **27**, a magnetic disk drive **28**, e.g., to read from or write to a removable disk **29**, and an optical disk drive **30**, e.g., for reading a CD-ROM disk **31** or to read from or write to other optical media. The hard disk drive **27**, magnetic disk drive **28**, and optical disk drive **30** are connected to the system bus **23** by a hard disk drive interface **32**, a magnetic disk drive interface **33**, and an optical drive interface **34**, respectively. The drives and their associated computer-readable media provide nonvolatile storage for the personal computer **20**. Although the description of computer-readable media above refers to a hard disk, a removable magnetic disk and a CD-ROM disk, it should be appreciated by those skilled in the art that other types of media which are readable by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, and the like, may also be used in the exemplary operating environment.

A number of software products may be stored in the drives and RAM **25**, including an operating system **35**, a software product **36**, such as Microsoft's "OFFICE XP" suite of application program modules, other software products **37**, and program data **38**. A user may enter commands and information into the personal computer **20** through a keyboard **40** and pointing device, such as a mouse **42**. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit **21** through a serial port interface **46** that is coupled to the system bus, but may be connected by other interfaces, such as a game port or a universal serial bus (USB). A monitor **47** or other type of display device is also connected to the system bus **23** via an interface, such as a video adapter **48**. In addition to the monitor, personal computers typically include other peripheral output devices (not shown), such as speakers or printers.

The personal computer **20** may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer **49**. The remote computer **49** may be a server, a router, a peer device or other common network node, and typically includes many or all of the elements described relative to the personal computer **20**, although only a memory storage device **50** has been illustrated in FIG. **1**. The logical connections depicted in FIG. **1** include a local area network (LAN) **51** and a wide area network (WAN) **52**. Such networking environments are commonplace in offices, enterprise-wide computer networks, Intranets and the Internet.

When used in a LAN networking environment, the personal computer **20** is connected to the LAN **51** through a network interface **53**. When used in a WAN networking environment, the personal computer **20** typically includes a modem **54** or other means for establishing communications over the WAN **52**, such as the Internet. The modem **54**, which may be internal or external, is connected to the system bus **23** via the serial port interface **46**. In a networked environment, program modules depicted relative to the personal computer **20**, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Radio Interface Layer

In one embodiment, the present invention, known as the Radio Interface Layer (RIL) comprises an API set which provides a level of abstraction between the radio on a cell phone and the software of the cell phone. The API set of RIL is based on the GSM AT interface as defined in GSM specifications 07.05 and 07.07. The API set provides access to functionality contained within a cellular telephone, such as a GSM or CDMA compatible telephone. The present invention allows applications running on an operating system in the cellular telephone to issue commands without knowledge of the underlying radio structure of the cellular telephone and specific knowledge of the GSM-type commands. For example, the present invention allows the applications to access phonebook entries, restrict access to data and functionality using passwords, access file and message storage, and perform many other functions.

The RIL is divided into a hardware-independent proxy layer, called by various software components, and a driver layer that is hardware-specific. It should be understood that an original equipment manufacturer (OEM) may replace the driver layer with their own layer containing implementation specific to their hardware. In a preferred embodiment, the RIL is a core component of a cellular telephone marketed by Microsoft Corporation of Redmond, Wash.

RIL Driver Layer

In a preferred embodiment, the Radio Interface Layer (RIL) driver layer is used to implement and roughly correspond to the commands, such as AT commands, specified by ETS 300 585, Digital cellular telecommunications system (Phase 2); Use of Data Terminal Equipment-Data Circuit terminating Equipment (DTE-DCE) interface for Short Messaging Service (SMS) and Cell Broadcast Service (CBS) (GSM 07.05), Fifth Edition, April 1997, and ETS 300 642, Digital cellular telecommunications system (Phase 2); AT command set for GSM Mobile Equipment (ME) (GSM 07.07 version 4.4.1), Fourth Edition, March 1999. The GSM specifications 07.05 and 07.07 are hereby incorporated by reference. Of course, the RIL driver may be used to implement and correspond to other command sets, such as CDMA commands, or a combination of several command sets.

OEMs may use the RIL driver of the preferred embodiment or tweak it if they would rather talk with their radio over private APIs instead of via AT commands (most likely for performance reasons).

Generally described, the RIL driver layer receives an RIL API call and causes the radio (i.e. the receiver/transmitter of the cellphone, PDA, etc.) to perform the function defined by the RIL API. In a preferred embodiment, the RIL driver will receive the RIL API call from a RIL proxy layer (described below). The RIL driver layer also processes notifications received from the radio and transmits them to the RIL proxy layer. In a preferred embodiment, the RIL driver layer is a dynamic link library (DLL) that is running as a device driver inside the process space of a device manager (the standard module that manages device drivers on the "WINDOWS CE" operating system). A Device Manager (device.exe) may be responsible for managing all the system drivers, including the RIL driver.

RIL Proxy Layer

In one embodiment, the RIL proxy layer comprises a layer that is called by various other layers of the core architecture, such as a TSP layer, an ExTAPI layer, and a SIM Manager using the platform specific commands of these core architectures. In a preferred embodiment, the proxy layer is a "WINDOWS CE" dynamic link library (DLL) that manages callback notifications and inter-process function calls into

the RIL driver layer. Modules that want to use the RIL simply link with this proxy DLL. The RIL proxy layer converts the core architecture specific commands into RIL API calls that will be understood by the RIL driver layer.

There are some important distinctions between the proxy and driver layers. In a preferred embodiment of the invention, a separate proxy instance is created for each module using the RIL proxy DLL. On the other hand, in a preferred embodiment of the invention, the RIL device driver is loaded only once and is shared amongst all proxy layer instances. In other words, a module using the RIL must be aware that only one radio module exists, even though it links to its own proxy DLL. In addition, the Device Manager's control of the RIL driver implies that the proxy and driver live in separate processes (i.e. different address spaces). However, the "WINDOWS CE" operating system exposes mechanisms allowing the proxy and driver layers to communicate without being concerned with the process boundaries.

Another important architectural property of the RIL is that almost all of the functions are asynchronous. When a module first registers with the RIL, it passes in two callback functions. One is used for unsolicited notifications, and the other is used for responses to function calls. For instance, when the phone receives a new incoming call, RIL will use the unsolicited notification callback to let each module know about the incoming call. Alternately, when a module calls RIL to obtain the signal strength, the function call immediately returns a response identifier. Shortly thereafter, RIL uses the function response callback to convey signal strength information to the module. To ensure that function response callbacks are correctly matched up with function calls, this callback structure also contains the same response identifier value returned by the original function call. This asynchronous architecture simplifies RIL implementation. If a module needs to call RIL functions in a synchronous manner, it will need to make the function call and block until it receives the function response callback.

Another architectural feature of the RIL is a virtual serial port (VSP). When an application makes a data connection, it retrieves a handle to a virtual serial port (not the real data stream between the RIL and the radio). This allows RIL to buffer and flow control the data stream so that control commands can be interjected. For example, suppose an application has set up a data connection and is browsing on the Internet. The virtual serial port allows RIL to interject control commands to check for things like signal strength, new SMS messages, etc. However, further suppose an application is receiving a facsimile. Due to strict timing issues in the case of a facsimile transmission, the RIL will enter a dedicated data mode where the application has full control of the virtual serial port. That is, RIL will not attempt to interject any commands in the data stream. It should be understood that the VSP is similar to other communication ports and typically only one application can have the handle to the VSP at one time.

Referring now to FIG. **2**, a block diagram illustrating an exemplary embodiment of an RIL in a cellphone **200** in accordance with an embodiment of the present invention will be described. The cellphone **200** comprises a SIM manager **205**, an emergency application **210**, a TAPI Service Provider (TSP) **215**, a WAP layer **220**, a SMS manager **225**, a data stack **230** and a VSP **250**.

The cellphone **200** further comprises a plurality of instances of the RIL proxy layer **235**. The RIL proxy layer **235** provides communications between applications (such as the SIM manager **205**, emergency application **210**, TSP **215**, WAP layer **220**, SMS manager **225**, and ExTAPI, among others) and a RIL driver layer **240**. The RIL driver layer **240** provides communications between the RIL proxy layer and the radio hardware **245**.

Scenarios

In terms of "using" the RIL (from the point of view of both an application designer and an OEM), the proxy and driver layers each expose a set of functions. For a program module to use the RIL, it only needs to use functions specified in the proxy header file and then link with the proxy DLL. The proxy DLL is implemented by calling functions specified in the driver header file. The driver header file is provided to OEMs and defines the functions that an OEM must implement. In one embodiment, the implementation will be hardware specific, so each OEM will be responsible for its own driver implementation. However, one or more reference implementations of the driver (source code included) may be provided to OEMs to help them with this step. If an OEM uses radio hardware that is supported by one of these reference implementations, it may not need to revise the RIL code.

Method for Processing Commands Using RIL

FIG. **3** is a flow diagram illustrating a method **300** for processing of commands using the radio interface layer (RIL) in accordance with an embodiment of the present invention. The method **300** begins at step **305** when a user calls a RIL API in the proxy layer.

At decision step **310**, it is determined whether the callee, i.e. the called process, is in the device.exe process space. If it is, then the method proceeds to step **315** where the driver APIs are called directly. If it is determined at decision step **310** that the callee is not in the device.exe process space then the method proceeds to step **320**.

At step **320**, input/output control (IOCTL) codes are used to send the appropriate information for the RIL API to the RIL driver running in a separate process space. At step **325**, the RIL driver informs the radio to take the action specified by the command of the RIL API. In a preferred embodiment, the RIL driver informs the radio to take action using an AT command interface, as defined in GSM specs (most prominently 07.05 and 07.07). However, sending AT commands may not be ideal for a given radio—perhaps an OEM has a separate private API set that they can use to perform the same functionality as a given AT command. If this is the case, the OEM may change the RIL driver to suit their needs. However, in a preferred embodiment, because the core architecture of the phone has been built on top of a set of RIL APIs that may be implemented via AT commands, it is not necessary for the OEM to substantially modify the RIL driver so long as the radio understands AT commands. However, due to different implementations of the AT interface, some minor modifications may be necessary.

The method then proceeds to step **330** where the RIL API returns with a unique ID generated by the RIL. It should be understood that after sending an AT command, a response from the radio unit is awaited. RIL APIs are designed to be asynchronous, so these APIs will return immediately, with a unique ID assigned to the call.

The method then proceeds to step **335** where a separate thread waits for responses from the radio unit.

The method then proceeds to step **340** where the RIL driver matches the response from the radio unit with the

unique ID generated earlier and the RIL driver sends the response to the appropriate calling process via a callback function.

It should also be understood that radio units can also send unsolicited notifications (for example, when the phone switches cellular towers). In this case, the RIL driver receives a notification from the radio and will broadcast a message to all users of the RIL layer who are interested in this class of notification.

For an example illustrating an implementation of method **300**, consider the following: the API RIL_ChangeLockingPassword is a RIL API that allows changing the password of a phone for various lock facilities. This API is modeled after the+CPWD AT command, defined in section 7.5 of GSM 07.07. The AT command to change a password requires a lock facility, the old password, and the new password. Accordingly, the API for RIL_ChangeLockingPassword appears as:

```
HRESULT RIL_ChangeLockingPassword(
HRIL hRil,
DWORD dwFacility,
LPCSTR lpszOldPassword,
LPCSTR lpszNewPassword
);
```

When the user application wants to change the locking password, it calls this API, typically indirectly via a TAPI layer or another layer. For example, the application may understand the TAPI command for changing a password and send this command to the TAPI layer. The TAPI layer will then make the appropriate RIL API call to the proxy layer. As part of the RIL API, a RIL handle must be provided (which is obtained from initializing RIL), a locking facility must be provided, the old password must be provided and the new password must be provided. For example, suppose a user application wishes to change the password used to lock the SIM card from "1234" to "5678". The user application (or an intermediate layer such as the TAPI layer) would make the following API call:

```
RIL_ChangeLockingPassword(hRIL, RIL_LOCKFACILITY_SIM, "1234", "5678");
```

If the calling process is not in the device.exe, these parameters will get bundled into a structure and passed via an IOCTL call, RIL_IOCTL_CHANGELOCKINGPASSWORD:

```
typedef struct changelockingpassword_params_tag {
    DWORD dwfacility;
    char szOldPassword[MAXLENGTH_PASSWORD];
    char szNewPassword[MAXLENGTH_PASSWORD];
}
```

CHANGELOCKINGPASSWORD_PARAMS, *LPCHANGELOCKINGPA SSOWRD_PARAMS;

The RIL driver will then take these constants and generate an AT command string as specified in GSM 07.07:

```
AT+CPWD=SC,1234,5678
```

Note that if an OEM were to change the RIL driver to call a private API to their radio instead of using an AT command, they would make their change at this point.

After sending this AT command (or private API) to the radio, the RIL driver returns, and RIL_ChangeLockingPassword returns. The radio has not yet processed the command at this point, so a unique ID is givenback to the user as the return value of this RIL API.

After processing the command, the radio module will return a success or error code (in this case, there will be a success code or a possibly descriptive error code such as "wrong password"). The radio gives this response to the RIL driver, which has a separate thread waiting for responses from the radio module. This response is then matched with the unique ID from the API call and sent via a callback function to the calling process. The calling process can then note whether the locking password was successfully changed or not and act accordingly.

Referring now to FIG. **4**, an example illustrating a method for an application to establish a voice call using RIL in accordance with an embodiment of the present invention will be described. It should be understood that establishing a voice call is only one of many functions that may be performed using the RIL APIs. The method outlined in FIG. **4** is illustrative of one of these functions (establishing a voice call). The method is implemented in telephone **400** that includes an application layer **405**, an ExTAPI layer **410**, a TAPI layer **415**, other core modules **420**, a TSP **425**, a radio interface layer (RIL) **430** in accordance with an embodiment of the invention, and OEM hardware **435**. It should be understood that the method does not describe the IOCTLs which will be understood by those skilled in the art to be present in a preferred embodiment. It should be further understood that the present invention may be implemented without the use of IOCTLs.

The method begins when application **405** calls the TAPI function: lineMakeCall (step **452**). The TAPI layer **415** calls the TSP **425** with the following function call: TSPI_lineMakeCall (step **454**). The TSP **425** calls the RIL with the following RIL function: RIL_Dial (step **456**). The RIL initiates the phone call by sending the corresponding command to the OEM hardware (e.g. radio): e.g. ATDT 555–1234 (step **458**). The TSP returns asynchronously a reply message to the TAPI layer indicating that the call has been initiated: LINE_REPLY message (step **460**).

The TAPI layer forwards the reply message (LINE_REPLY) to the application (step **462**). When the OEM hardware detects that a connection to the number has been made, then it sends a CONNECT response to the RIL (step **464**). The RIL sends a message (RIL_NOTIFY_CONNECT) to the TSP indicating that a connection has been made (step **466**). The TSP sends a state change message (LINE_CALLSTATE) to the TAPI layer (step **468**). The TAPI layer forwards the state change message (LINE_CALLSTATE) to the application **405** (step **470**).

When the application **405** wishes to drop the telephone call, it calls the TAPI layer with a hang-up request: (lineDrop) (step **472**). The TAPI layer forwards the TSP the hang-up request: (TSPI_lineDrop) (step **474**). The TSP handler passes the hang-up request to the RIL: (RIL_Hangup) (step **476**). The RIL passes the hang-up request to the OEM hardware (e.g. ATH) (step **478**).

Features

The following table describes some of the features that may be implemented using an embodiment of the present invention and a brief description of these features.

Name	Description
Network Services	Operator selection, signal status, available operators, etc.
Call Control	Dial, Hangup, Hold, Transfer, etc.
Supplemental Services	Call waiting, call forwarding, call meter, etc.
SMS	Send, receive, SMSC selection, etc.
Data Connection	Virtual serial port, bearer service types, transparent data/online mode change
Security Functions	Locking, unlocking, passwords, etc.
Interface Functions	Initialization, notification registration, etc.
Phonebook Functions	Accessing the SIM phonebooks
SIM SMS Functions	Accessing SMS messages from the SIM
GPRS Functions	Selecting profiles, attaching, activating, etc.
HSCSD Functions	Managing channel allocations
SIM Record Access	Access individual files on the SIM
SIM Toolkit 2.0 Support	Engaging in a SIM toolkit session

Structure Listing

This section describes the "data structures" passed as parameters to some RIL APIs and returned with some RIL notifications in an embodiment of the present invention.

Structure	Comments
Network Service Structures	
RILSUBSCRIBERINFO	Defines an assigned phone number and parameter associated with it
RILOPERATORNAMES	Defines the long, short, and numeric format of a single operator
RILOPERATORINFO	Defines a network operator
Call Control Structures	
RILCALLINFO	Defines each call instance (active, on hold, in conference, etc.)
RILRINGINFO	Indicates the type of incoming call
Supplemental Service Structures	
RILCALLFORWARDSETTINGS	Defines call forwarding
RILCALLWAITINGINFO	Information about an incoming call
RILCALLERIDSETTINGS	Defines caller ID
RILHIDEIDSETTINGS	Defines how to hide your phone number when calling someone else
RILDIALEDIDSETTINGS	Defines the true number that was dialed
RILCLOSEDGROUPSETTINGS	Defines closed user group settings
RILREMOTEPARTYINFO	Structure used for CallerID and DialedID notifications
Voice Structures	
RILGAININFO	Defines audio gain for the transmit and receive channels
RILAUDIODEVICEINFO	Defines transmit and receive audio devices
Messaging Structures	
RILMSGSERVICEINFO	Messaging settings such as storage locations and usage info
RILMSGDCS	Data coding scheme
RILMSGCONFIG	Messaging configuration
RILMESSAGE	An actual message
RILMESSAGEINFO	Contains an RILMESSAGE along with additional info
RILMSGSTORAGEINFO	Information about a message storage location
Data Service Structures	
RILCALLHSCSDINFO	Defines HSCSD parameters for the current call
RILHSCSDINFO	Defines HSCSD parameters
RILDATACOMPINFO	Defines parameters for data compression
RILERRORCORRECTIONINFO	Defines parameters for error correction
RILBEARERSVCINFO	Defines the current data communication settings
RILRLPINFO	Defines Radio Link Protocol (RLP) parameters for non-transparent data calls
RILCONNECTINFO	Defines connect information on a data call
RILSERIALPORTSTATS	Defines statistics of the virtual serial port
RILSERVICEINFO	Defines parameters of the data connection
RILSUPSERVICEDATA	Defines elements of a USSD message
Capability Structures	
RILCAPSBEARERSVC	Bearer service capabilities
RILCAPSDIALSVC	Dial capabilities
RILCAPSHSCSD	HSCSD capabilities
RILCAPSLOCKINGPWD LENGTH	Locking password length capabilities
RILCAPSMSGMEMORY LOCATIONS	Message storage location capabilities
RILCAPSRLP	RLP capabilities
SIM Toolkit Structures	
RILSIMCMDPARAMETERS	Elements of a SIM command
RILSIMRESPONSE	Elements of a SIM command response
Miscellaneous Structures	
RILEQUIPMENTINFO	Defines miscellaneous (generally static) properties of radio module
RILPHONEBOOKINFO	Defines the state of the phonebook
RILPHONEBOOKENTRY	Defines an entry in the phonebook
RILCOSTINFO	Defines cost information for the current operator's rate
RILSIGNALQUALITY	Defines the current signal quality
RILADDRESS	A phone number
RILSUBADDRESS	More detailed information about a phone number
RILCELLTOWERINFO	Information about the currently registered cell tower
RILRANGE	Defines a min/max range

Notification Listing

This section lists some of the unsolicited RIL notifications that get passed to the notification callback. Note that these notifications differ from the ones passed to the response

callback as responses to earlier issued function calls. These notifications have been categorized for convenience. These notifications are in one embodiment of the invention and are not meant to limit the invention. dwCode is the numeric ID identifying the notification and lpData is the additional data returned with the notification.

DwCode	lpData	Comments
RIL_NOTIFY_REGSTATUS CHANGED	(RIL_REGSTAT_ *) Constant	Sent with change in registration status
RIL_NOTIFY_CALLMETER	dwNewCallMeter	Call meter has changed
RIL_NOTIFY_CALLMETER MAXREACHED	<NULL>	Maximum call meter has been reached
	Call Control Notifications	
RIL_NOTIFY_RING	RILRINGINFO Structure	Incoming call
RIL_NOTIFY_CONNECT	RILCONNECTINFO Structure	Call connected
RIL_NOTIFY_DISCONNECT	RIL_DISCINIT_* Constant	Call disconnected
RIL_NOTIFY_DATASVC NEGOTIATED	RILSERVICEINFO Structure	Data call service has been negotiated
RIL_NOTIFY_CALLSTATE CHANGED	<NULL>	Call state of one or more calls may have changed
RIL_NOTIFY_EMERGENCY MODEENTERED	<NULL>	RIL has entered emergency mode
RIL_NOTIFY_EMERGENCY MODEEXITED	<NULL>	RIL has exited emergency mode
RIL_NOTIFY_EMERGENCY HANGUP	<NULL>	Existing calls (if any) were hung up for emergency mode
RIL_NOTIFY_HSCSDPARAMS NEGOTIATED	RILCALLHSCSDINFO Structure	HSCSD parameters for a call have been negotiated
	Supplemental Service Notifications	
RIL_NOTIFY_CALLERID	(RILREMOTEPARTYINFO *)	The remote address of the incoming call
RIL_NOTIFY_DIALEDID	(RILREMOTEPARTYINFO *)	The destination address of the outgoing call
RIL_NOTIFY_CALLWAITING	(RILCALLWAITINGINFO *)	Call waiting notification
RIL_NOTIFY_SUPSERVICE DATA	(RILSUPSERVICEDATA*)	Incoming USSD message
	Messaging Notifications	
RIL_NOTIFY_MESSAGE	(RILMESSAGE *)	Indicates a new message
RIL_NOTIFY_BCMESSAGE	(RILMESSAGE *)	Indicates a new broadcast message
RIL_NOTIFY_STATUS MESSAGE	(RILMESSAGE *)	Indicates a new status message
RIL_NOTIFY_MSGSTORED	(dwIndex)	Indicates a message has been stored
RIL_NOTIFY_MSGDELETED	(dwIndex)	Indicates a message has been deleted
RIL_NOTIFY_MSGSTORAGE CHANGED	RILMSGSTORAGEINFO Structure	One of the message storage locations has been changed
	Phonebook Notifications	
RIL_NOTIFY_PHONEBOOK ENTRYSTORED	dwIndex	Phonebook entry has been added
RIL_NOTIFY_PHONEBOOK ENTRYDELETED	dwIndex	Phonebook entry has been deleted
RIL_NOTIFY_PHONEBOOK STORAGECHANGED	(RIL_PBLOC *) Constant	Phonebook storage location has been changed

-continued

DwCode	IpData	Comments
	SIM Toolkit Notifications	
RIL_NOTIFY_SIMTOOLKIT CMD	dwByteCount	Proactive SIM command received
RIL_NOTIFY_SIMTOOLKIT CALLSETUP	dwRedialTimeout	Proactive SIM command to setup a call
RIL_NOTIFY_SIMTOOLKIT EVENT	dwByteCount	Toolkit command was handled by the radio or radio sent a toolkit response to the SIM
	Miscellaneous Notifications	
RIL_NOTIFY_SMSNOT ACCESSIBLE	<NULL>	Sim has been removed or has failed to respond
RIL_NOTIFY_DTMFSIGNAL	(char*)	A DTMF signal has been detected

Function Listing

This section lists some of the RIL functions broken down by group. Each entry denotes the function name, and a brief description. Where applicable, the corresponding GSM AT command is included.

Function	GSM	Comments
	Network Service Functions	
RIL_GetSubscriberNumbers	+CNUM	Gets list of assigned phone numbers
RIL_GetOperatorList	+COPS	Gets a list of available operators
RIL_GetPreferredOperatorList	+CPOL	Gets a list of preferred operators
RIL_AddPreferredOperator	+CPOL	Adds to the list of preferred operators
RIL_RemovePreferredOperator	+CPOL	Removes from the list of preferred operators
RIL_GetCurrentOperator	+COPS	Gets the operator currently registered
RIL_RegisterOnNetwork	+COPS	Register with a particular operator
RIL_UnregisterFromNetwork	+COPS	Unregister current operator
RIL_GetRegistrationStatus	+CREG	Gets registration status
	Call Control Functions	
RIL_Dial	D +FCLASS	Dials a number
RIL_Answer	A	Answers an incoming call
RIL_Hangup	H	Sets operator information
RIL_SendDTMF	+VTS	Sends DTMF tones (e.g. during a call)
RIL_GetDTMFDuration	+VTD	Gets tone duration options
RIL_SetDTMFDuration	+VTD	Sets tone duration options
RIL_SetDTMFMonitoring		Turns on/off DTMF monitoring

-continued

Function	GSM	Comments
RIL_GetCallList	+CLCC	Retrieves list of active calls and their status
RIL_ManageCalls	+CHLD	Changes call status (hold, conference, etc)
RIL_TransferCall	+CTFR	Explicitly transfers a call
RIL_GetLineStatus	+CPAS	Gets line status
	Supplemental Service Functions	
RIL_SetCallerIDSettings	+CLIP	CallerID settings
RIL_GetHideIDSettings	+CLIR	Hides own number from recipient
RIL_SetHideIDStatus	+CLIR	Hides own number from recipient
RIL_GetDialedIDSettings	+COLP	Dialed number on an outgoing call
RIL_SetDialedIDSettings	+COLP	Dialed number on an outgoing call
RIL_GetClosedGroupSettings	+CCUG	Closed user group settings
RIL_SetClosedGroupSettings	+CCUG	Closed user group settings
RIL_GetCallForwardSettings	+CCFC	Call forward settings
RIL_AddCallForwarding	+CCFC	Add a number to the call forwarding list
RIL_RemoveCallForwarding	+CCFC	Remove a number from the call forwarding list
RIL_SetCallForwardStatus	+CCFC	Enable/disable call forwarding
RIL_GetCallWaitingSettings	+CCWA	Call waiting settings
RIL_SetCallWaitingStatus	+CCWA	Call waiting settings
	Voice Functions	
RIL_GetAudioGain	+VGR +VGT	Gets receive gain of the audio device
RIL_SetAudioGain	+VGR +VGT	Sets receive gain of the audio device
RIL_GetAudioDevices	+VGS	List connected audio devices (mic, speaker, etc)
RIL_SetAudioDevices	+VGS	Sets connected audio devices (mic, speaker, etc)
RIL_GetAudioMuting	+CMUT	Gets muting state
RIL_SetAudioMuting	+CMUT	Sets muting state
	Messaging Functions	
RIL_GetMsgServiceOptions	+CSMS +CPMS +CMGF +CESP	Gets messaging service options
RIL_SetMsgServiceOptions	+CSMS +CPMS +CMGF +CESP	Sets messaging service options
RIL_GetMsgConfig	+CSCA +CSMP +CSDH +CSCB	Gets message configuration options
RIL_SetMsgConfig	+CSCA +CSMP +CSDH +CSCB	Sets message configuration options
RIL_RestoreMsgConifg	+CRES	Restores messaging settings
RIL_SaveMsgConfig	+CSAS	Saves messaging settings
RIL_GetMsgList	+CMGL	Lists all messages
RIL_ReadMsg	+CMGR +CMGD	Read (optionally delete) a message
RIL_DeleteMsg	+CMGD	Delete a message
RIL_WriteMsg	+CMGW +CMGS +CMMS	Writes (optionally send) a message

-continued

Function	GSM	Comments
RIL_SendMsg	+CMGS +CMSS +CMMS	Send a message
RIL_SendStoredMsg	+CMGS +CMSS +CMMS	Send a message from a storage location
RIL_SendMsgAcknowledgement	+CMGS +CMSS +CMMS	Send a message ACK when requested by an incoming message
	Data Service Functions	
RIL_GetSerialPortHandle		Gets a virtual serial port handle
RIL_GetSerialPortStatistics		Gets statistics on the virtual serial port handle
RIL_GetHSCSDOptions	+CHSD +CHDT +CHSN +CHSC +CHSR	Get settings for circuit switched data calls
RIL_SetHSCSDOptions	+CHSD +CHDT +CHSN +CHSC +CHSR	Set settings for circuit switched data calls
RIL_GetDataCompression	+DS	Gets data compression options
RIL_SetDataCompression	+DS	Sets data compression options
RIL_GetErrorCorrection	+EX	Gets error correction options
RIL_SetErrorCorrection	+EX	Sets error correction options
RIL_GetBearerServiceOptions		
RIL_SetBearerServiceOptions		
RIL_GetRLPOptions		Gets radio link protocol options
RIL_SetRLPOptions		Sets radio link protocol options
RIL_CancelSupServiceDataSession	+CUSD	Cancel a USSD session
RIL_SendSupServiceData	+CUSD	Send a USSD message
	Security Functions	
RIL_GetUserIdentity	+CIMI	Retrieve the customer's mobile identity
RIL_UnlockPhone	+CPIN	Sends a pending password
RIL_ChangeCallBarringPassword	+CPIN +CPWD	Changes the call barring password
RIL_ChangeLockingPassword	+CPIN +CPWD	Changes the locking password
RIL_GetPhoneLockedState	+CPIN	Gets phone lock status
RIL_GetCallBarringStatus	+CLCK	Gets call barring status
RIL_SetCallBarringStatus	+CLCK	Sets call barring status
RIL_GetLockingStatus	+CLCK	Gets locking status
RIL_SetLockingStatus	+CLCK	Sets locking status
	Interface Functions	
RIL_Initialize		Registers RIL proxy with RIL driver
RIL_Deinitialize		Unregisters RIL proxy from RIL driver

-continued

Function	GSM	Comments
RIL_InitializeEmergency		Registers an emergency application
RIL_DeinitializeEmergency		Unregisters an emergency application
RIL_EnableNotifications		Sets which notification classes to receive
RIL_DisableNotifications		Disables notification classes from being sent
	Phonebook Functions	
RIL_GetPhonebookOptions	+CPBS	Gets the phonebook options
RIL_SetPhonebookOptions	+CPBS	Sets the phonebook location
RIL_DeletePhonebookEntry	+CPBW	Deletes a phonebook entry
RIL_ReadPhonebookEntries	+CPBR	Get phonebook entry
RIL_WritePhonebookEntry	+CPBW	Writes a phonebook entry
	SIM Toolkit Functions	
RIL_FetchSimToolkitCmd	+CSIM	Retrieves a proactive toolkit command
RIL_GetSimToolkitProfile		Retrieves a current profile for a profile download
RIL_SetSimToolkitProfile		Sets current profile for a profile download
RIL_SendSimToolkitCmd Response	+CSIM	Sends a response to a proactive toolkit command
RIL_SendSimToolkitEnvelope Cmd	+CSIM	Sends an envelope command to the SIM
RIL_TerminateSimToolkit Session	+CSIM	Terminates a toolkit session
	Miscellaneous Functions	
RIL_GetEquipmentInfo	+CGMI GMI +CGMM GMM +CGMR GMR +CGSN GSN	Retrieves information about the phone equipment
RIL_GetEquipmentState	+CFUN	Manages phone state (power management)
RIL_SetEquipmentState	+CFUN	Manages phone state (power management)
RIL_SendSimCmd	+CSIM	Sends unrestricted commands directly to a SIM
RIL_SendRestrictedSimCmd	+CRSM	Sends a restricted set of commands directly to a SIM
RIL_ResetTerminal	Z & F	Resets all terminal parameters to defaults
RIL_GetCostInfo	+CAOC +CPUC	Retrieves advice of charge information
RIL_SetCostInfo	+CAOC +CPUC	Sets advice of charge information

-continued

Function	GSM	Comments
RIL_GetSignalQuality	+CSQ	Gets signal quality
RIL_GetDevCaps		Retrieves the capabilities of the radio device
RIL_DevSpecific		Developer specific command
RIL_GetCellTowerInfo	+CREG	Gets info about the currently used cell tower

It should be understood from the foregoing description that the RIL proxy layer is hardware-independent. In contrast, it should be understood that in different embodiments, the RIL driver layer is hardware-specific. However, in one embodiment, a sample GSM implementation of the RIL driver is provided to function with generic GSM hardware (although, in practice, some modifications will probably be needed for almost any GSM system currently in existence because the GSM specifications may be interpreted and implemented slightly differently by different OEMs).

It should be also understood from the foregoing description, that the present invention allows software applications to function on RIL-compatible phones independently of the hardware or the cellular network being used. For example, changing from a GSM to a CDMA network would only require replacing the RIL driver layer and the rest of the phone would work as it did in the GSM network.

It should be understood from the foregoing description that the purpose of the RIL is to provide access to cellular functionality for any component in the phone, PDA, etc. Without the RIL, each component (TAPI, SIM manager, SMS manager, etc.) of the phone would have to understand how to communicate to the radio directly. Because it would be difficult for hardware manufacturers to implement a TAPI driver, a SMS driver, a SIM driver, etc., the RIL was created to sit between the radio and the TAPI driver, the SMS driver, the SIM driver, etc.

It should be also be understood from the foregoing description, that because the RIL proxy is hardware-independent, RIL provides a platform for third party software developers. With the well-designed APIs and interfaces of the RIL of the present invention, a third party software developer may write his code once and have it work on all devices containing an implementation of RIL, such as telephones, PDAs, etc. Moreover, the software developer may use the well-defined telephony commands such as TAPI without worrying about whether the underlying device is using cellular technology, voice over IP, etc.

It should be understood that one of the objectives of RIL is to ease the integration process of software components with an OEM's hardware components. To realize this, a single layer handles all communication between the core modules and an OEM's radio hardware. The single RIL allows software components to be designed without having to worry about differences in underlying hardware. It also allows OEMs to integrate the software components with their radio hardware by implementing a single set of functions.

It should be understood that the foregoing description includes many implementation details that should not limit the scope of the present invention. For example, instead of using a proxy layer and a driver layer, the present invention may be implemented as a single abstraction layer between a telephony radio and a computer. The applications on the computer may communicate with the abstraction layer using top-level APIs. On the other hand, the telephony radio would respond to commands received from the abstraction layer. Because the difficulties of implementing specific modules to understand different protocols such as TAPI, ExTAPI, SMS, etc. is accomplished by the RIL itself, the present invention eases the implementation difficulties radio manufacturers often have. Moreover, radio manufacturers no longer have to worry about receiving and keeping track of calls from multiple client applications because all of these functions are handled by the RIL. Software application developers do not need to worry about the underlying hardware of a mobile device. Software applications may be easily written to work with RIL because the applications use well-known top-level APIs which are sent to the RIL. The RIL will then perform appropriate processing of these top-level APIs and, if necessary, send the appropriate command to the radio to perform a specific action.

Other Supported Configurations

It should also be understood from the foregoing description, that the present invention may be used with cellular telephones as well as other devices, such as hand-held PDA devices. Some of these other devices may not have a permanent radio module(s). Certain changes known to those skilled in the art may be necessary to implement the invention in a device without a permanent radio module(s). Specifically, the invention must support Removable Compact Flash (CF)/PCMCIA radio modules that support circuit-switched cellular network connections.

Listed below are some possible device configurations:

Configuration 1: Cellular telephone

The device has a built in radio-module. It does not have any expansion slots that support CF or PCMCIA cards. Therefore the built-in radio module is guaranteed to be always present and no alternative form of cellular communications is permitted.

Configuration 2: PDA with PCMCIA/CF support

The device does not have a built in radio-module. However, it does contain a CF and/or PCMCIA expansion slot(s). In a preferred embodiment, the invention requires that a supported Radio module be inserted into the CF or PCMCIA slot.

Configuration 3: PDA with built in Radio and PCMCIA/CF support

The device has a built in radio module. It can be assumed that this radio module will always be present. Potentially, one can insert other devices (including radio modules) into any available expansion slot (PCMCIA, USB, Bluetooth, etc,).

The devices described above may also require a few additions and modifications to the API set as described below in an illustrative embodiment:

PDA Support API additions

Error Codes:

RIL_E_RADIONOTPRESENT

Fails the RIL calls because there isn't a radio present in the system

RIL_E_RADIOREMOVED

Fails the RIL calls which were in the process of being executed because the radio was removed

PDA Notifications

RIL_NCLASS_RADIOSTATE

Radio State notifications (RIL_NCLASS_RADIOSTATE)

Notification Radio State Constants

RIL_NOTIFY_RADIOPRESENT

Notification corresponding to when the radio is inserted and the RIL Driver is ready to accept commands

RIL_NOTIFY_RADIONOTPRESENT

Notification corresponding to when the radio is removed and the RIL Driver is unloaded.

Additional Unrelated Notifications

RIL_NOTIFY_RADIOOFF

For SetEquipmentState TxandRX off command

RIL_NOTIFY_RADIOON

For SetEquipmentState TxandRX on command

Attached as Appendix A is a list of the RIL APIs of a preferred embodiment of the present invention. These APIs are provided as examples only and should not limit the present invention.

We claim:

1. An abstraction layer for interfacing a computer to a telephony radio, comprising:

a set of application programming interfaces (APIs) for abstracting out multiple radio technologies without knowledge of the telephony radio or cellular network, wherein the set of APIs correspond to call control functions, wherein the abstraction layer comprises a proxy layer and a driver layer, wherein when the proxy layer receives a call at a first interface to one of the set of APIs, the proxy layer transforms the API call to a command understood by the driver layer and sends the command to the driver layer at a second interface, and wherein the driver layer receives the command at the second interface and determines at least one standard telephony radio command corresponding to the called API and sends the telephony radio command to the telephony radio at a third interface, and wherein the proxy layer is hardware independent and the driver is hardware specific.

2. The abstraction layer of claim **1** wherein the telephony radio is one of a plurality of telephony radios which operates based on the standard telephony radio commands.

3. The abstraction layer of claim **1** wherein the set of APIs further correspond to short messaging system functions.

4. The abstraction layer of claim **3** wherein the set of APIs further correspond to network service functions.

5. The abstraction layer of claim **4** wherein the set of APIs further correspond to data connection functions.

6. The abstraction layer of claim **5** wherein the set of APIs further correspond to interface functions.

7. A radio interface layer of a telephone for facilitating communications between an application program module and a radio, comprising:

a proxy layer for communicating with the application program module at a first interface and a driver layer at a second interface, wherein the proxy layer provides an API on the first interface for receiving application program calls to perform a particular function and wherein the proxy layer transforms the API calls to an input/output control (IOCTL) code and sends the IOCTL code to the driver layer at the second interface;

wherein the driver layer communicates with the proxy layer at the second interface and the radio at a third interface, the driver layer receiving an IOCTL code at the second interface and transforming the IOCTL code into a command understood by the radio to perform the particular function and sending the radio command at the third interfaces; and

wherein the proxy layer is hardware independent and the driver layer is hardware specific.

8. The radio interface layer of claim **7** wherein the driver layer further receives communications from the radio indicating that the particular function has been performed and wherein the driver layer sends a success code to the proxy layer indicating that the particular function has been performed.

9. A method for processing commands in a telephone comprising a proxy layer, a driver layer, an application and a radio, the method comprising the steps of:

causing the application to call a radio interface layer (RIL) API in the proxy layer at a first interface, wherein the RIL API is associated with an action to be performed by the radio;

causing the proxy layer to translate the RIL API into IOCTL codes;

sending the IOCTL codes to the driver layer at a second interface;

translating the IOCTL codes to a command corresponding to the action, wherein the command will be understood by the radio;

sending the command to the radio at a third interface; and

wherein the proxy layer is hardware independent and the driver layer is hardware specific.

10. The method of claim **9** wherein the command is an AT command.

11. The method of claim **9** wherein the command is one of a private API set defined by the radio manufacturer.

12. The method of claim **9** further comprising the step of generating in the RIL driver layer a unique ID associated with the RIL API.

13. The method of claim **12** further comprising the step of waiting for a response from the radio, and when received, calling back the calling application with the response and the unique ID returned from the call.

14. The method of claim **13** wherein the RIL driver matches the response from the radio with the unique ID and the RIL driver sends the response to the calling process via a callback function.

15. A method of communicating between a module and a radio comprising:

(a) generating a radio interface layer (RIL) API call at one of a plurality of modules to perform a specific action;

(b) sending the RIL API call to a proxy at a first interface;

(c) at the proxy, converting the RIL API call to a command understood by a radio driver;

(d) transmitting the radio driver command from the proxy to the radio driver at a second interface;

(e) transmitting a radio command from the radio driver to the radio at a third interface;

(f) performing the specific action at the radio; and

wherein the proxy is hardware indepentent and the driver is hardware specific.

16. The method of claim further **15** comprising:

(g) in response to successfully performing the specific action, sending a success code from the driver to the proxy and from the proxy to the one of the plurality of modules that generated the RIL API.

17. The method of claim **16** wherein the RIL API, command and success code are associated with an identifier linking them together and linking them to the one of the plurality of modules that generated the RIL API call and wherein the radio driver receives the success code, and, using the identifier, matches the success code with the one of the plurality of modules that generated the RIL API call and sends the success code to the one of the plurality of modules that generated the RIL API call.

18. The method of claim **17** further comprising the step of:

(h) generating a notification at the radio in response to detecting data that needs to be reported to one of the plurality of modules;

(i) sending the notification to the radio driver.

19. The method of claim **18** further comprising the step of:

(j) sending the notification from the radio driver to the proxy.

20. The method of claim **19** further comprising the step of:

(k) sending the notification from the proxy to at least one of the plurality of modules.

21. The method of claim **18** wherein the data that needs to be reported comprises an incoming phone call to the radio.

22. The method of claim **18** wherein the data that needs to be reported comprises a signal strength change in the radio.

23. The method of claim **18** wherein the one of a plurality of modules is a TSP.

24. The method of claim **18** wherein the one of a plurality of modules is a SIM manager.

25. The method of claim **18** wherein the one of a plurality of modules is an emergency application for generating emergency calls.

26. The method of claim **18** wherein the one of a plurality of modules is a WAP layer.

27. The method of claim **18** wherein the one of a plurality of modules is a TAPI interface.

28. The method of claim **18** wherein the one of a plurality of modules is an ExTAPI interface.

29. The method of claim **18** wherein the one of a plurality of modules is connected to an application program module and receives instructions from the application program module to generate the RIL API call.

30. The method of claim **29** wherein the instructions provided by the application program module comprise instructions defined by the one of a plurality of modules and wherein the instructions are converted to the Rib API calls by the one of a plurality of modules.

* * * * *

US006909910B2

(12) **United States Patent**
Pappalardo et al.

(10) **Patent No.: US 6,909,910 B2**
(45) **Date of Patent: Jun. 21, 2005**

(54) **METHOD AND SYSTEM FOR MANAGING CHANGES TO A CONTACT DATABASE**

(75) Inventors: **Susan Elizabeth Pappalardo**, Kirkland, WA (US); **Jason William Fuller**, Bellevue, WA (US); **Peter G. Chin**, Seattle, WA (US); **Jessica Dale Tenenbaum**, Seattle, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 447 days.

(21) Appl. No.: **10/061,763**

(22) Filed: **Feb. 1, 2002**

(65) **Prior Publication Data**

US 2003/0148790 A1 Aug. 7, 2003

(51) **Int. Cl.**[7] **H04M 1/64**

(52) **U.S. Cl.** **455/558**; 455/418; 455/414.1; 707/100; 707/101; 707/102

(58) **Field of Search** 455/558, 414.1, 455/418, 564, 565, 566, 567; 707/102, 101, 100; 70/100

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,943,055 A * 8/1999 Sylvan 715/839
6,741,994 B1 * 5/2004 Kang et al. 707/102
2003/0083046 A1 * 5/2003 Mathis 455/412
2004/0066920 A1 * 4/2004 Vandermeijden 379/88.19

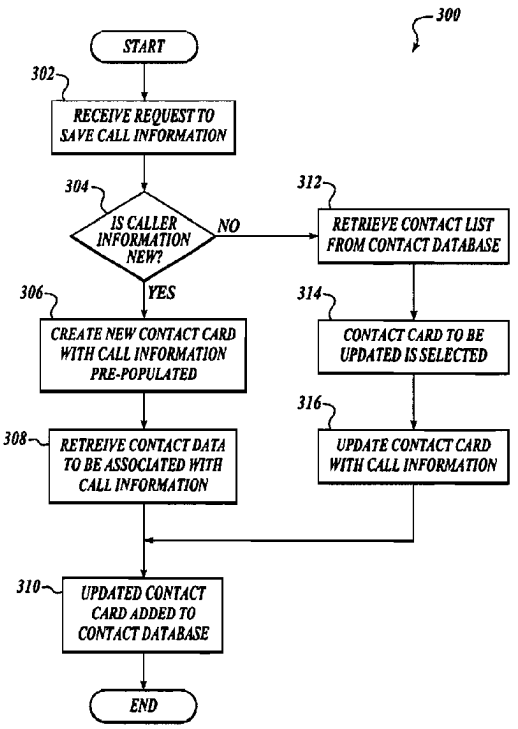
* cited by examiner

Primary Examiner—Nick Corsaro
Assistant Examiner—Alan T. Gantt
(74) *Attorney, Agent, or Firm*—Merchant & Gould P.C.

(57) **ABSTRACT**

Described is a system and method for updating a contact and adding a new contact from a call log in a communications device. The system includes a contact manager that is directed towards creating and updating call contact cards in a contact database with information retrieved from call logs of phone calls made to or from the communications device. In one embodiment, information is pre-populated into a predetermined data field of the contact card, thereby reducing workload to a user. The method includes determining if a request is for updating an existing contact card or for adding a new contact card to the contact database. The update or addition is made with information retrieved from call logs. Call information is pre-populated into a predetermined data field of the contact card, when it is determined that the request is to add a new contact card to the contact database.

10 Claims, 8 Drawing Sheets



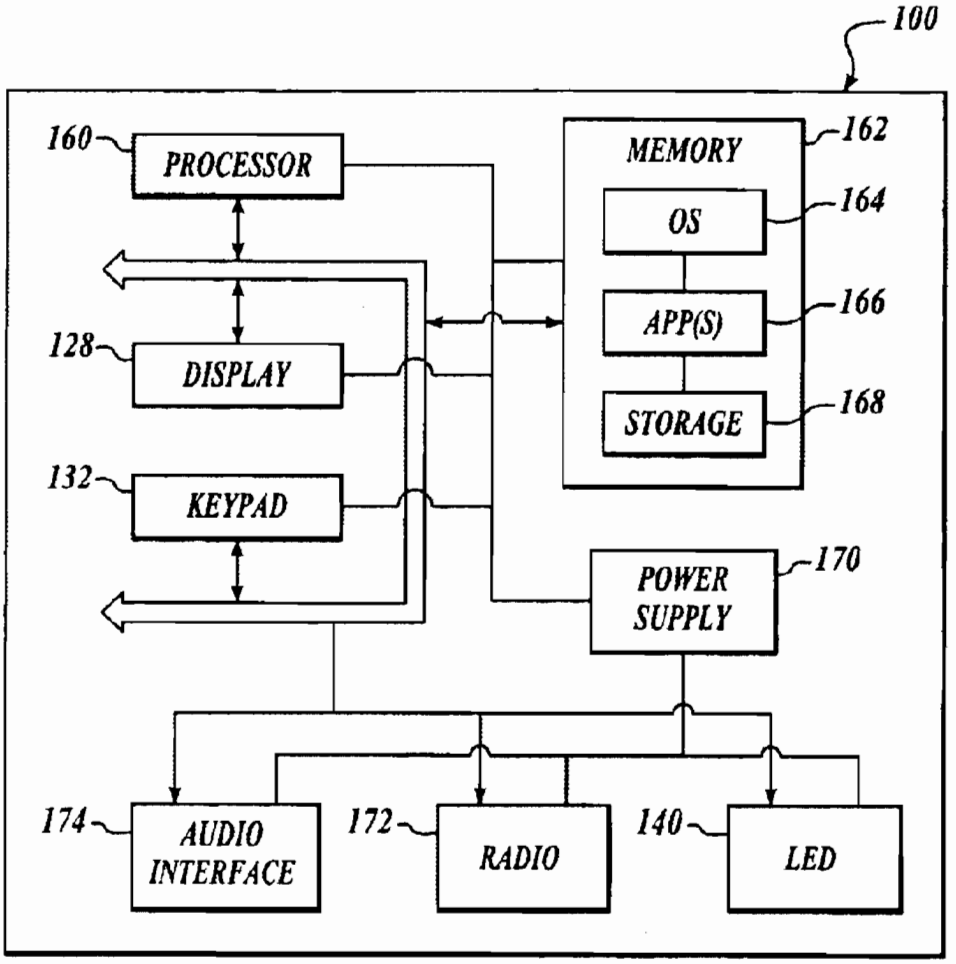

100
160
PROCESSOR
MEMORY
162
OS
164
APP(S)
166
128
DISPLAY
STORAGE
168
132
KEYPAD
POWER SUPPLY
170
174
AUDIO INTERFACE
172
RADIO
140
LED

FIG. 1

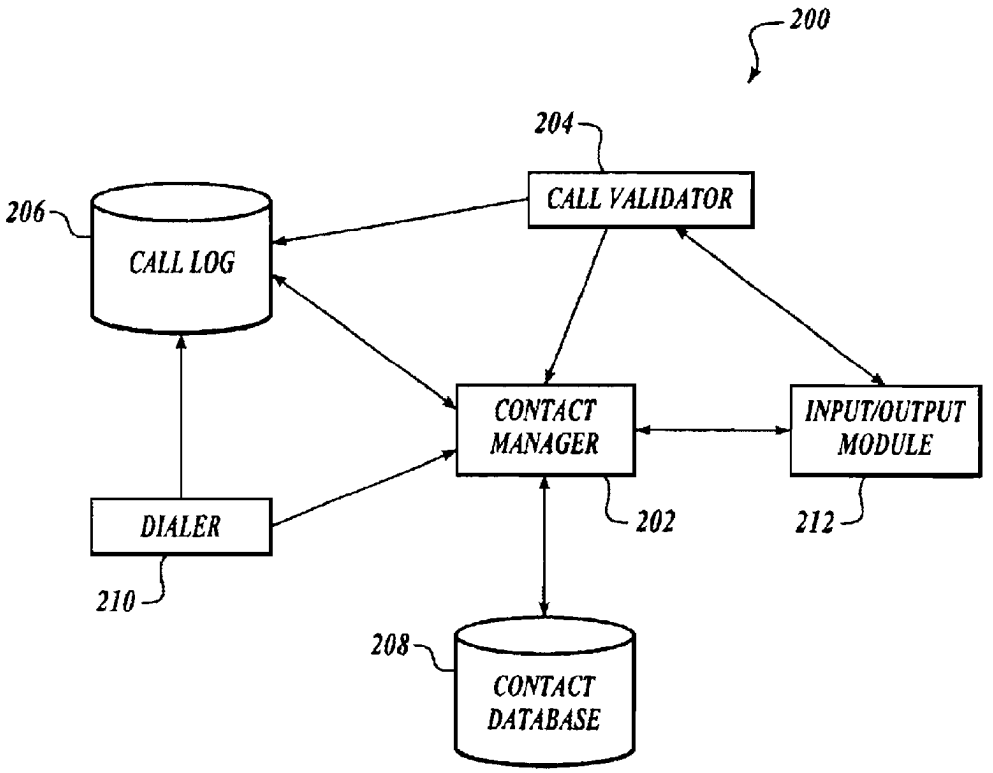

200
204
CALL VALIDATOR
206
CALL LOG
CONTACT MANAGER
INPUT/OUTPUT MODULE
202
212
DIALER
210
208
CONTACT DATABASE

FIG. 2

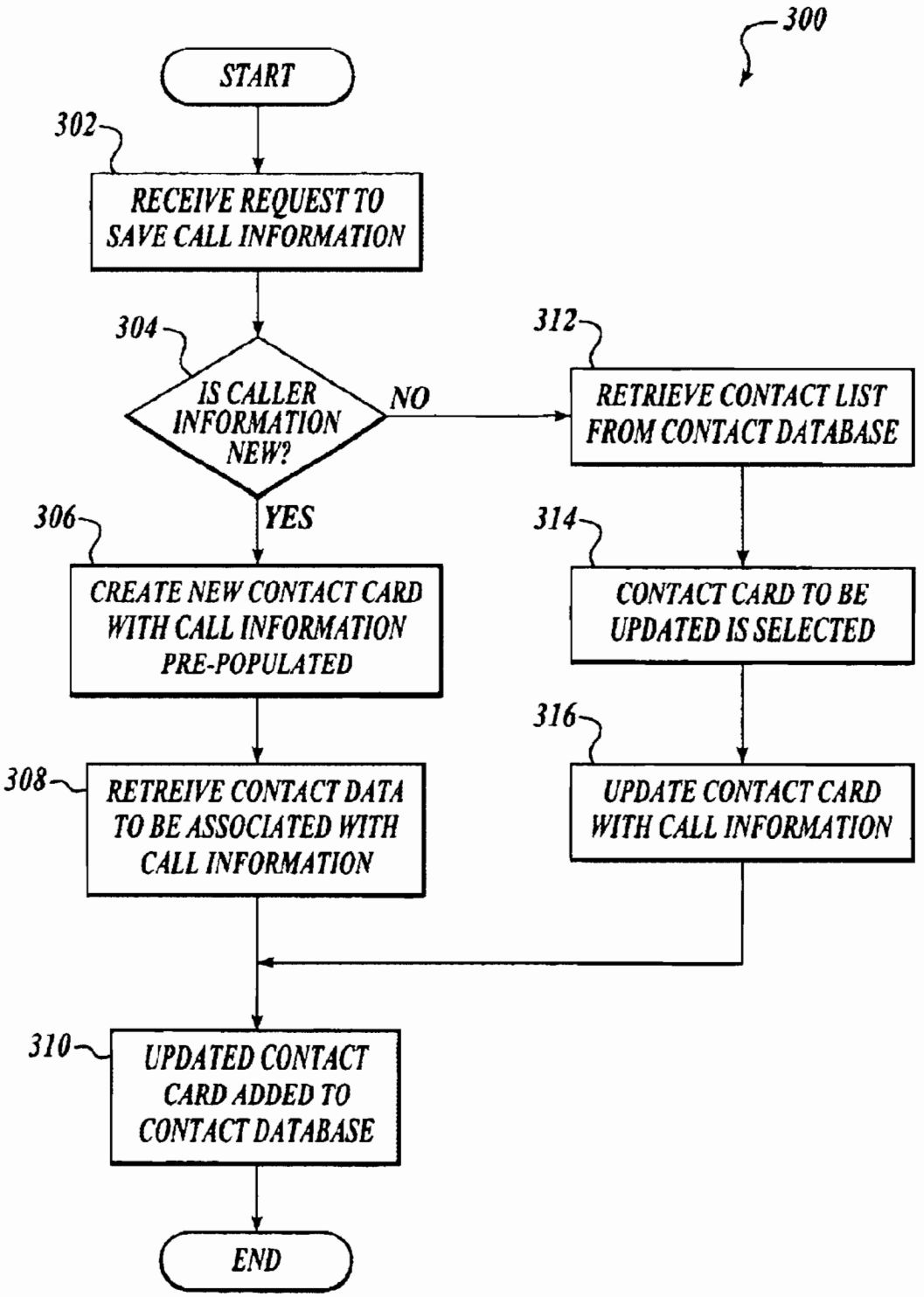

300
START
302
RECEIVE REQUEST TO SAVE CALL INFORMATION
304
IS CALLER INFORMATION NEW?
NO
YES
312
RETRIEVE CONTACT LIST FROM CONTACT DATABASE
306
CREATE NEW CONTACT CARD WITH CALL INFORMATION PRE-POPULATED
314
CONTACT CARD TO BE UPDATED IS SELECTED
316
UPDATE CONTACT CARD WITH CALL INFORMATION
308
RETREIVE CONTACT DATA TO BE ASSOCIATED WITH CALL INFORMATION
310
UPDATED CONTACT CARD ADDED TO CONTACT DATABASE
END

FIG. 3

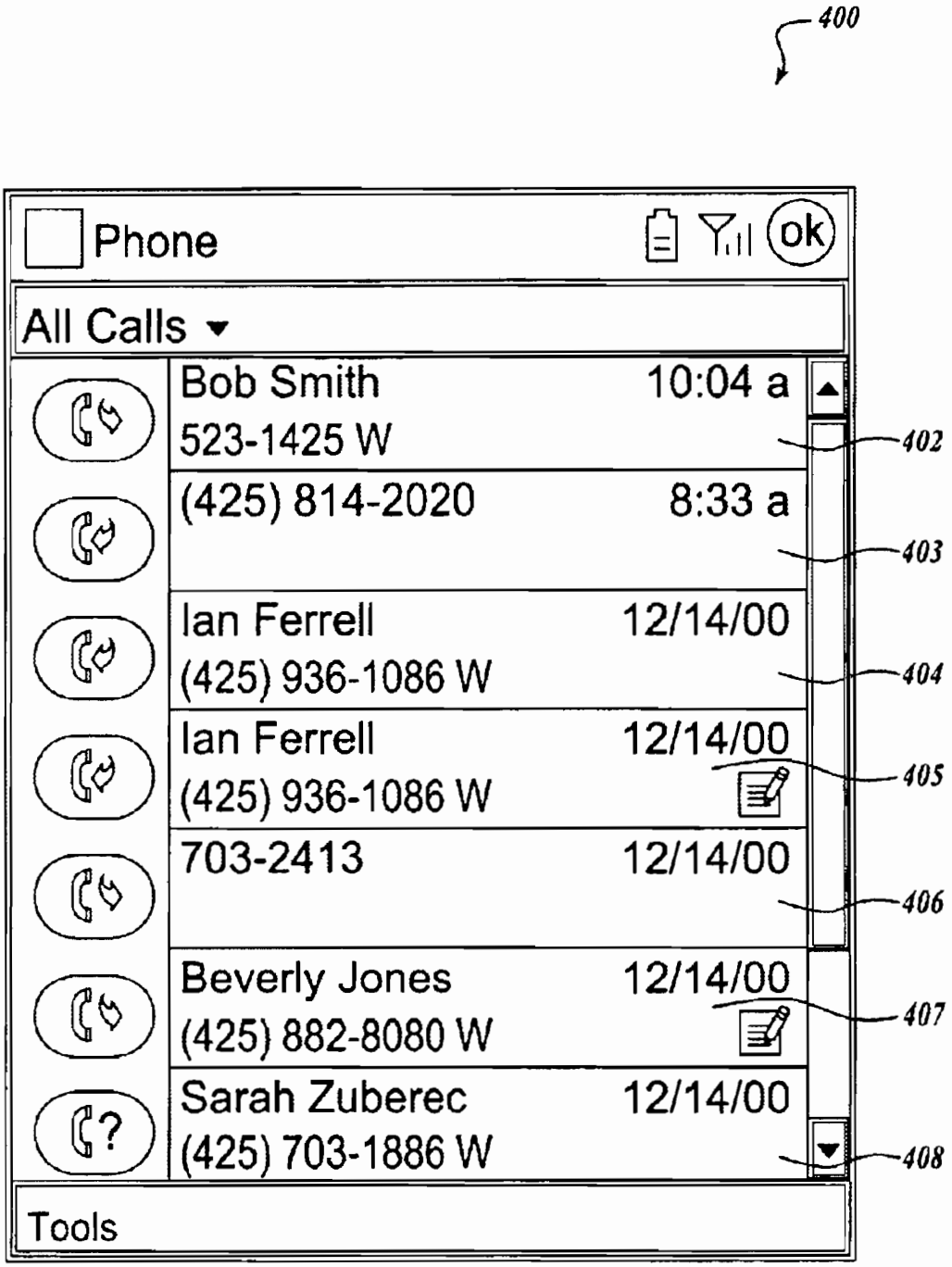

400
Phone
ok
All Calls
Bob Smith 10:04 a
523-1425 W
(425) 814-2020 8:33 a
Ian Ferrell 12/14/00
(425) 936-1086 W
Ian Ferrell 12/14/00
(425) 936-1086 W
703-2413 12/14/00
Beverly Jones 12/14/00
(425) 882-8080 W
Sarah Zuberec 12/14/00
(425) 703-1886 W
Tools
402
403
404
405
406
407
408

FIG. 4

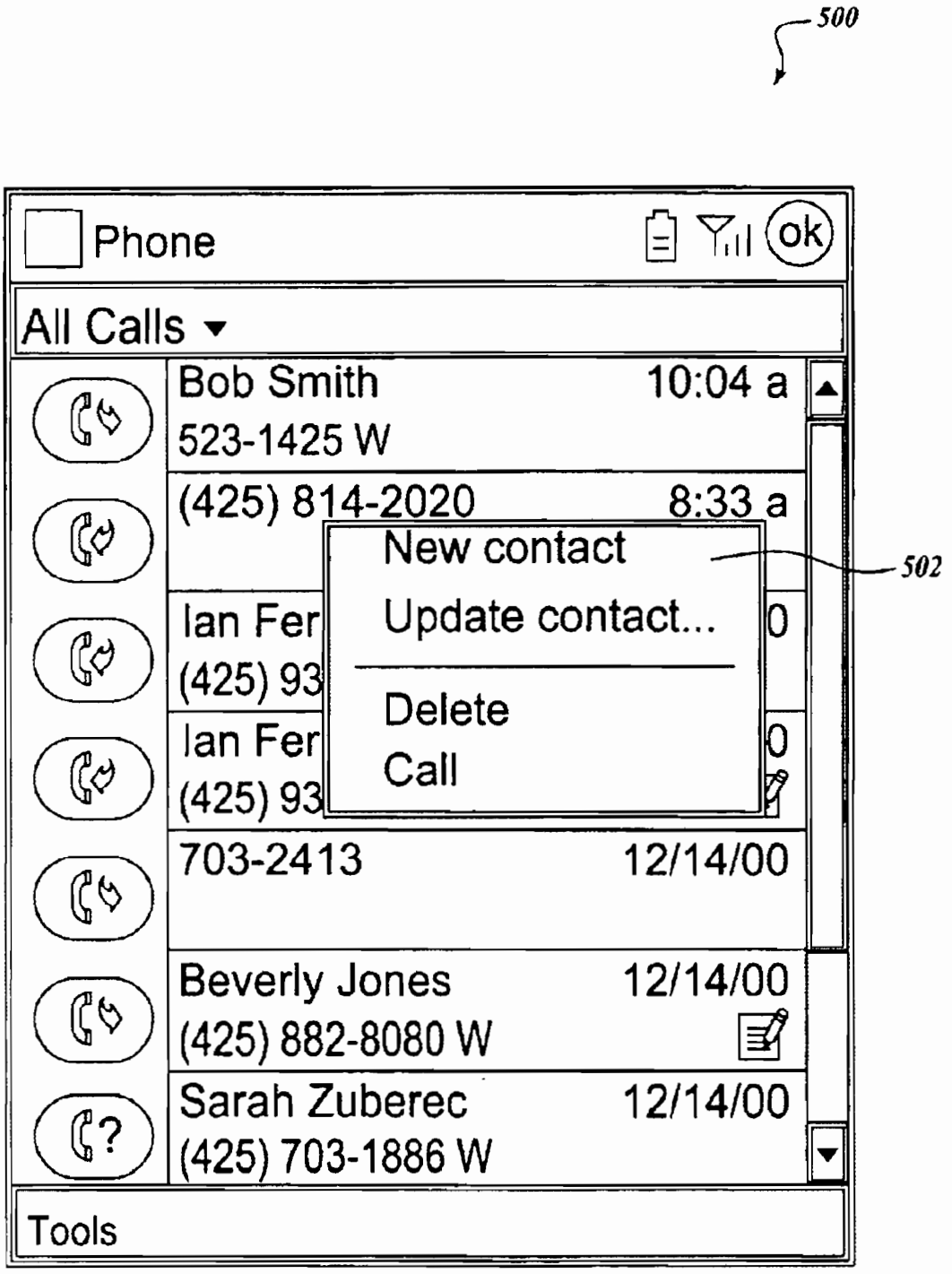

500
Phone
ok
All Calls
Bob Smith
10:04 a
523-1425 W
(425) 814-2020
8:33 a
New contact
Update contact...
Delete
Call
502
Ian Fer
(425) 93
Ian Fer
(425) 93
703-2413
12/14/00
Beverly Jones
12/14/00
(425) 882-8080 W
Sarah Zuberec
12/14/00
(425) 703-1886 W
Tools

FIG. 5

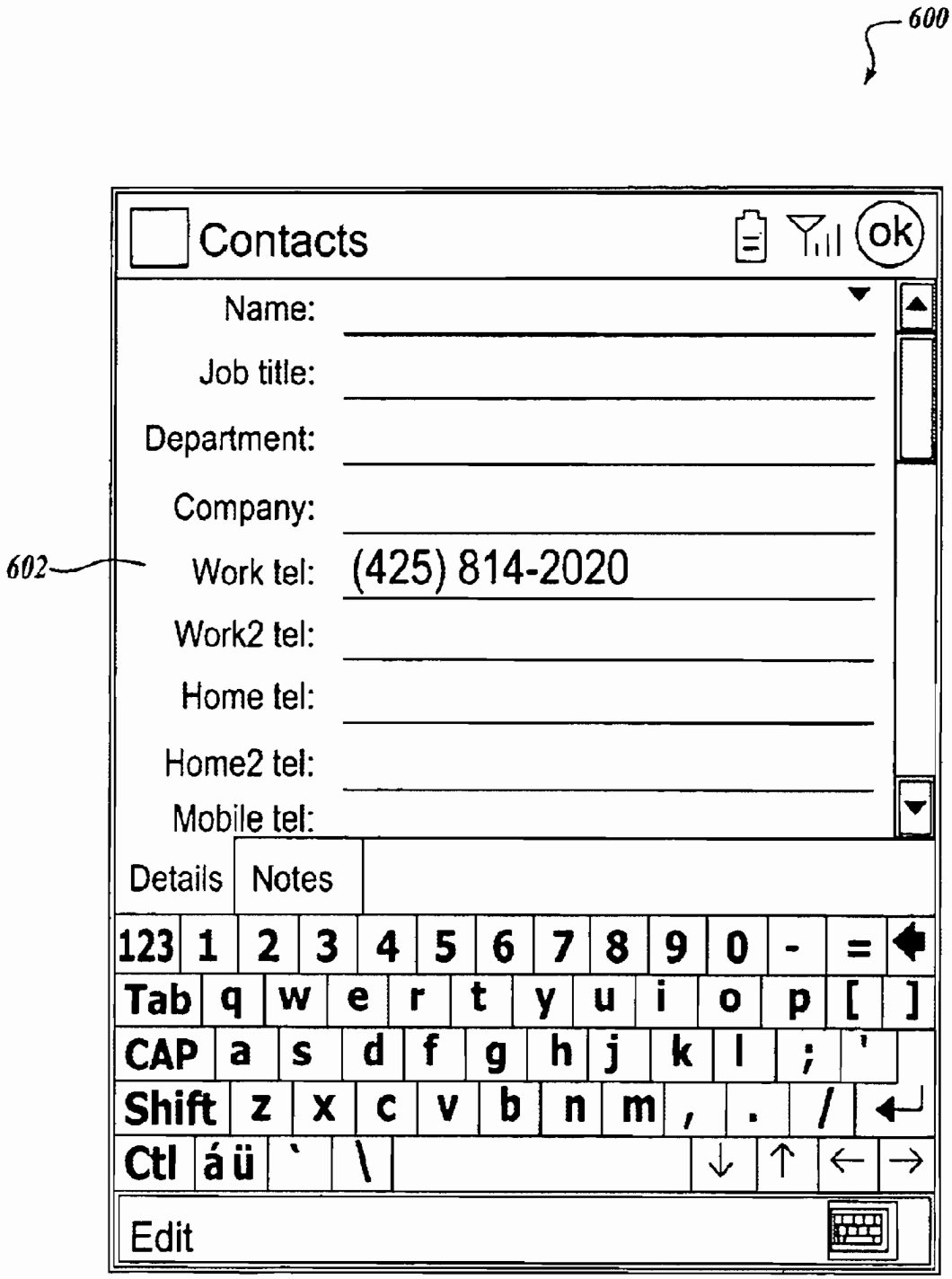

600
Contacts
ok
Name:
Job title:
Department:
Company:
602
Work tel:
(425) 814-2020
Work2 tel:
Home tel:
Home2 tel:
Mobile tel:
Details
Notes
123 1 2 3 4 5 6 7 8 9 0 - =
Tab q w e r t y u i o p [ ]
CAP a s d f g h j k l ; '
Shift z x c v b n m , . /
Ctl áü ` \
Edit

FIG. 6

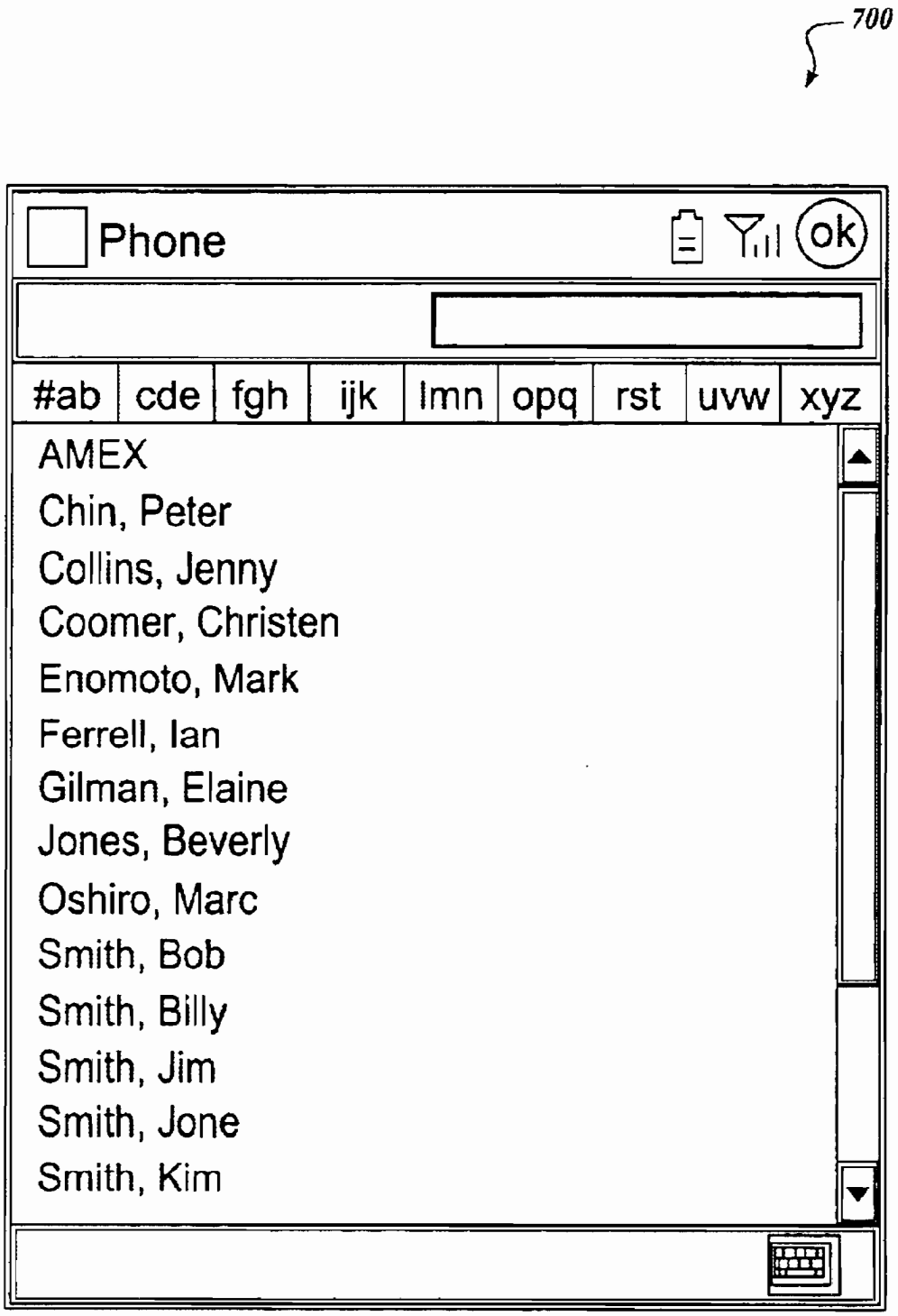

700
Phone
ok
#ab
cde
fgh
ijk
lmn
opq
rst
uvw
xyz
AMEX
Chin, Peter
Collins, Jenny
Coomer, Christen
Enomoto, Mark
Ferrell, Ian
Gilman, Elaine
Jones, Beverly
Oshiro, Marc
Smith, Bob
Smith, Billy
Smith, Jim
Smith, Jone
Smith, Kim

FIG. 7

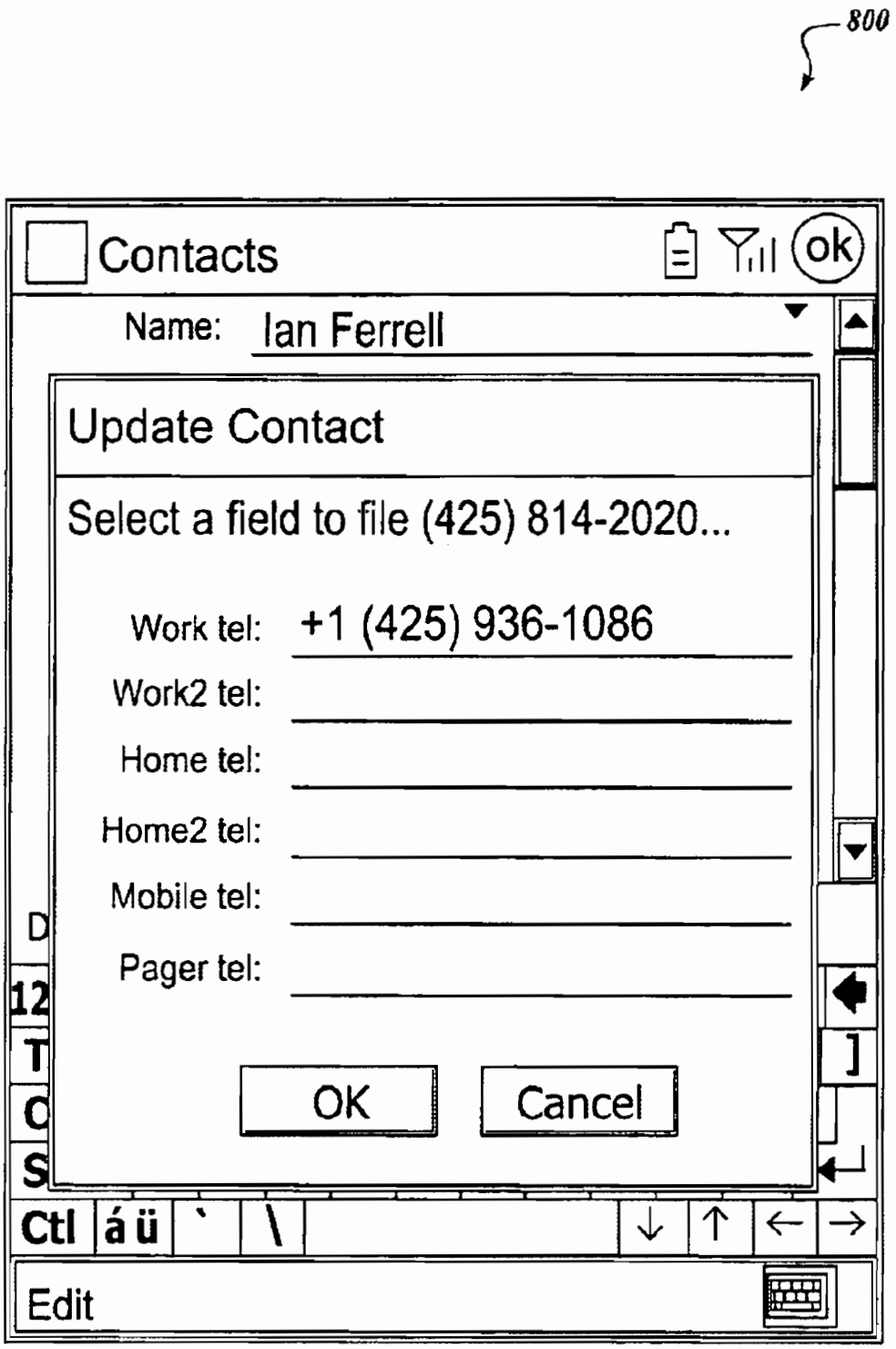

800
Contacts
ok
Name: Ian Ferrell
Update Contact
Select a field to file (425) 814-2020...
Work tel: +1 (425) 936-1086
Work2 tel:
Home tel:
Home2 tel:
Mobile tel:
Pager tel:
OK
Cancel
Ctl
á ü
Edit

FIG. 8

METHOD AND SYSTEM FOR MANAGING CHANGES TO A CONTACT DATABASE

FIELD OF THE INVENTION

The present invention relates generally to mobile computing, and more particularly to updating a contact database within a mobile computing device.

BACKGROUND OF THE INVENTION

Manufacturers have recently observed an increased demand by businesses and consumers for multi-functional mobile communications devices. In response, manufacturers have added to mobile communication devices, such as mobile telephones, a wealth of applications and services. For example, many mobile telephones include such features as graphical displays to support web access, contact lists, and e-mail services, as well as other non-voice features.

Recently, some manufacturers have responded by combining the features of personal digital assistants (PDAs) with the features of mobile telephone devices. However, while the features of PDAs and mobile telephone devices have been physically combined into a single mobile device, many of the application programs continue to operate independently from each other. Moreover, much of the data associated with one application remains inaccessible by another application, often resulting in increased frustration and workload for the consumer.

SUMMARY OF THE INVENTION

This summary of the invention section is intended to introduce the reader to aspects of the invention and is not a complete description of the invention. Particular aspects of the invention are pointed out in other sections herein below and the invention is set forth in the appended claims, which alone demarcate its scope.

The present invention is directed towards providing a method and system for updating a contact and adding a new contact from call logs in a mobile communications device. The system includes a contact manager that is directed towards creating and updating contact cards in a contact database with information retrieved from call logs of phone calls made to or from the communications device. The method includes determining if a request is for updating an existing contact card or for adding a new contact card to the contact database. The update or addition is then made with information retrieved from call logs. According to one aspect of the present invention, information is pre-populated into a predetermined data field of the contact card, thereby reducing workload to a user.

In accordance with one aspect of the present invention, a computer-implemented method is directed towards managing changes to a contact database. The method includes receiving a request to update a contact card stored in the contact database with call information related to a phone call, retrieving a contact list of contact cards stored in the contact database, and receiving a selection of a contact card to be updated within the contact list. The method further includes updating the selected contact card with the call information related to the phone call, and replacing the existing contact card in the contact database with the updated contact card.

In another aspect of the present invention, a computer-implemented method is directed to managing changes to a contact database. The method includes receiving a request to create a new contact card in the contact database with call information related to a phone call. The method pre-populates a predetermined data field of the new contact card with call information; receives contact data to be associated with the call information; modifies a data field in the new contact card with the received contact data; and updates the contact database with the modified contact card. The predetermined data field includes at least one of a home phone number, a work phone number, and a mobile phone number.

In accordance with yet another aspect of the present invention, a computer-readable medium is encoded with computer-executable components. The components include a contact database, a call log, and a contact manager. The contact database is configured to store contact cards, wherein contact cards include data fields configured to contain contact information. The call log is configured to record incoming and outgoing phone calls, wherein each phone call corresponds to a call entry in the call log, and each call entry includes call information. The contact manager is coupled to the contact database and the call log, wherein the contact manager is arranged to update at least one data field in the contact card in the contact database with call information from a call entry in the call log in response to a user instruction.

Still another aspect of the present invention is directed towards a mobile communications device for creating a new contact card in a contact database. The device includes a contact database, a call log, and a contact manager. The contact database is configured to store contact cards, wherein contact cards include data fields for containing contact information. The call log is configured to log incoming and outgoing phone calls, wherein each phone call corresponds to a call entry in the call log, and each call entry includes call information. The contact manager is arranged to create the new contact card in the contact database with call information from a call entry in the call log in response to a user instruction.

In accordance with yet another aspect of the present invention, a device includes a graphical user interface and a user selection interface mechanism, and a method of activating a selection for changing a contact database. The method includes displaying a list of call entries in a call log of phone calls, displaying a context menu in response to a user selection of an entry in the call log, and displaying a plurality of options in the context menu, one of the options being to update a contact card in the contact database with call information from the selected entry in the call log.

Moreover, in yet another aspect of the present invention, a computer-readable medium is encoded with a plurality of data structures comprising a first data structure and a second data structure. The first data structure includes a plurality of call entries, each call entry describing call information about a phone call. The second data structure includes a plurality of contact cards, each contact card being associated with a contact. At least one contact card includes updated call information from a call entry in the first data structure.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing aspects and many of the attendant advantages of the present invention will become more readily appreciated as the same becomes better understood by reference to the following detailed description, when taken in conjunction with the accompanying drawings, wherein:

FIG. **1** is a functional block diagram illustrating an embodiment of an exemplary system for practicing the present invention;

FIG. **2** is a functional block diagram of an embodiment of components in communications device **100** of FIG. **1** for managing changes to a contact database;

FIG. **3** is a flow diagram generally showing an embodiment of a process for updating and adding a contact to the contact database;

FIG. **4** is an illustrative screen shot of an embodiment of a user-interface (UI) for a call log of all calls;

FIG. **5** is an illustrative screen shot of an embodiment of a UI for a context menu for updating or adding a contact card to the contact database;

FIG. **6** is an illustrative screen shot of an embodiment of a UI for a contact card for a new contact with a pre-populated phone number entry;

FIG. **7** is an illustrative screen shot of an embodiment of a UI for a contact list from the contact database; and

FIG. **8** is an illustrative screen shot of an embodiment of a UI for an update contact card, in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention now will be described more fully hereinafter "with reference to the accompanying drawings, which form a part hereof, and which show, by way of illustration, specific exemplary embodiments by which the invention may be practiced. This invention may, however, be embodied in many different forms and should not be construed as limited to the embodiments set forth herein; rather, these embodiments are provided so that this disclosure will be thorough and complete, and will fully convey the scope of the invention to those skilled in the art. Among other things, the present invention may be embodied as methods or devices. Accordingly, the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment or an embodiment combining software and hardware aspects. The following detailed description is, therefore, not to be taken in a limiting sense.

Throughout the specification, the term "connected" means a direct connection between the things that are connected, without any intermediary devices or components. The term "coupled," means a direct connection between the things that are connected, or an indirect connection through one or more either passive or active intermediary devices or components. The meaning of "a," "an," and "the" include plural references. The meaning of "in" includes "in" and "on."

Briefly stated, the present invention provides a computer-implemented system and method for updating a contact database in a mobile communications device from information, such as a phone number, in a call log. The method includes determining whether the phone number is new or already exists in the contact database, and providing an opportunity for updating or adding the phone number to a contact card in the contact database.

Illustrative Operating Environment

FIG. **1** is a functional block diagram illustrating an embodiment of an exemplary communications device **100** for practicing the present invention. In one embodiment of the present invention communications device **100** is implemented as a mobile communications device, such as an integrated personal digital assistant (PDA) and wireless phone.

As shown in the figure, communications device **100** includes processor **160**, memory **162**, display **128**, and keypad **132**. Memory **162** generally includes both volatile memory (e.g., RAM) and non-volatile memory (e.g., ROM, Flash Memory, or the like). Communications device **100** includes an operating system **164**, such as the Windows CE operating system from Microsoft Corporation or other such operating system, which is resident in memory **162** and executes on processor **160**. Keypad **132** may be a push button numeric dialing pad (such as on a typical telephone), a multi-key keyboard (such as a conventional keyboard). Display **128** may be a liquid crystal display, or any other type of display commonly used in mobile computing devices. For example, display **128** may be touch-sensitive, and would then also act as an input device.

One or more application programs **166** are loaded into memory **162** and run on the operating system **164**. Examples of application programs include phone dialer programs, contact manager, email programs, scheduling programs, word processing programs, spreadsheet programs, and so forth. Communications device **100** also includes non-volatile storage **168** within memory **162**. Non-volatile storage **168** may be used to store persistent information which should not be lost if the communications device **100** is powered down. The application programs **166** may use and store information in storage **168**, such as e-mail or other messages used by an e-mail application, contact information used by the contact manager, appointment information used by a scheduling program, documents used by a word processing application, and the like. A synchronization application may also reside on communications device **100** and is programmed to interact with a corresponding synchronization application resident on a host computer to keep the information stored in storage **168** synchronized with corresponding information stored at the host computer.

Communications device **100** also includes power supply **170**, which may be implemented as one or more batteries. Power supply **170** might further include an external power source, such as an AC adapter or a powered docking cradle that supplements or recharges the batteries.

Communications device **100** is also shown with two types of external notification mechanisms: LED **140** and audio interface **174**. These devices may be directly coupled to power supply **170** so that when activated, they remain on for a duration dictated by the notification mechanism even though processor **160** and other components might shut down to conserve battery power. LED **140** may be programmed to remain on indefinitely until the user takes action to indicate the powered-on status of the device. Audio interface **174** is used to provide audible signals to and receive audible signals from the user. For example, audio interface **174** may be coupled to a speaker for providing audible output and to a microphone for receiving audible input, such as to facilitate a telephone conversation.

Communications device **100** also includes radio **172** that performs the function of transmitting and receiving radio frequency communications. Radio **172** facilitates wireless connectivity between communications device **100** and the outside world, via a communications carrier or service provider. Transmissions to and from radio **172** are conducted under control of operating system **164**. In other words, communications received by radio **172** may be disseminated to application programs **166** via operating system **164**, and vice versa.

Radio **172** allows communications device **100** to communicate with other computing devices, such as over a network. Radio **172** is one example of communication media. Communication media may typically be embodied by computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as a

carrier wave or other transport mechanism, and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. The term computer readable media as used herein includes both storage media and communication media.

FIG. **2** is a functional block diagram of an embodiment showing components for managing changes to a contact database, in accordance with the present invention. Contact Management System **200** in FIG. **2** is an illustration of only one example of components and is not intended to suggest any limitation as to the scope of use or functionality of the present invention.

Contact Management System **200** may be described in the general context of computer-executable instructions, such as application programs **166**, being executed in communications device **100**, described in conjunction with FIG. **1**. The invention may also be performed by a combination of hardware and computer-executable instructions.

As shown in the figure, Contact Management System **200** includes contact manager **202**, contact database **208**, call validator **204**, dialer **210**, call log **206**, and input/output module **212**. Each of these components may communicate with each other either directly or indirectly by passing messages through operating system **164**.

Contact manager **202** is a software component that may be implemented within operating system **164** or as one of application programs **166** illustrated in FIG. **1**. Contact manager **202** is configured to interact with other components in Contact Management System **200** to receive and manage changes to contact database **208**.

Contact database **208** includes information related to a contact, such as people, businesses, and the like, with which communications device **100** may communicate. Contact database **208** may include contact information organized for each contact by separating the information of the contact into associated data fields, within a contact card. These data fields may include several phone numbers, e-mail addresses, pager number, contact name, and other call information related to the contact.

Contact database **208** is configured to receive instructions to create new contact cards and to update existing contact cards. The instructions may include the information to add to an existing contact card or with which to create a new contact card.

Call validator **204** includes software components that are configured to receive and decode call information about a caller of an incoming phone call. Call information may include a caller's phone number, the name of the caller, date and time of call, and the like. A communications carrier or service provider may provide call information to the callee of the incoming phone call when such information is available. When an incoming phone call is received (even if the phone call is not answered), call validator **204** may provide call information to call log **206** and contact manager **202**. Call validator **204** may also provide call information to input/output module **212** for viewing through display **128** of FIG. **1**.

Dialer **210** includes software components that are arranged to direct outgoing phone calls for communications device **100**. Dialer **210** also may be in communications with and take directions from operating system **164** of FIG. **1**. For example, operating system **164** may direct dialer **210** to make an outgoing phone call through radio **172** to a particular phone number provided by dialer **210**.

Dialer **210** may make the outgoing phone call by requesting a wireless phone connection from a communications carrier or service provider through radio **172** (FIG. **1**). When the wireless connection is established, dialer **210** may provide call information to call log **206**, and contact manager **202**. Call information may include phone number, date, time, call duration, and the like.

Call log **206** includes a record of calls made to and from communications device **100**. In one embodiment call log **206** is implemented as a database. Each call is recorded as a call entry in call log **206**. A call entry in call log **206** includes information about a particular phone call, such as phone number and whether the call was an incoming, outgoing, or missed call. Call entries may also include dates and times associated with the phone call.

Contact manager **202** may communicate with call log **206** to retrieve a call entry, or a category of call entries, such as missed calls. Contact manager **202** may incorporate additional contact information, such as a name, stored in contact database **208** that is associated with the call entry.

Furthermore, contact manager **202** may employ information in a call entry, in dialer **210**, or in call validator **204** to create new contact cards or update existing contact cards within contact database **208**.

Contact manager **202** also may provide the information to input/output module **212** for viewing through display **128** of FIG. **1**. Briefly referring to FIG. **4** is an illustrative screen shot of an embodiment of a user-interface (UI) **400** for a call log, in accordance with the present invention. Included in UI **400** are call entry display field(s) **402–408**. A typical call entry display field **402**, may include name, phone number, date, time, and an indicator of whether the call entry is an incoming, outgoing, or missed call. It will be apparent that UI **400** may display only a subset of calls made to and from communications device **100** that the user has not deleted from call log **206**. For example, UI **400** may display only outgoing, missed, or incoming calls, or a combination of the above. Moreover, while UI **400** illustrates only seven call entry display fields, UI **400** is not so limited, and more or less call entry display fields may be displayed, without departing from the scope or spirit of the present invention.

Input/output module **212** is a software component that is configured to provide input data to contact manager **202** that is received from hardware and other software components of communications device **100**. Input/output module **212** also provides output data from contact manager **202** to other components of communications device **100**. For example, keyboard **132** may inform contact manager **202** through input/output module **212** that an outgoing phone number has been entered by the user.

Generalized Operation

FIG. **3** is a flowchart generally showing an embodiment of an exemplary process **300** for updating and adding a contact to the contact database, in accordance with the present invention. Process **300** may be employed by contact manager **202** illustrated in FIG. **2**.

FIGS. **5–8** are employed to provide illustrative examples to further aid in illustrating the flowchart.

It will be understood that each block of the flowchart illustration, and combinations of blocks in the flowchart illustration, can be implemented by computer program instructions. These program instructions may be provided to a processor to produce a machine, such that the instructions, which execute on the processor, create means for implementing the actions specified in the flowchart block or

blocks. The computer program instructions may be executed by a processor to cause a series of operational steps to be performed by the processor to produce a computer implemented process such that the instructions, which execute on the processor provide steps for implementing the actions specified in the flowchart block or blocks.

Accordingly, blocks of the flowchart illustration support combinations of means for performing the specified actions, combinations of steps for performing the specified actions and program instruction means for performing the specified actions. It will also be understood that each block of the flowchart illustration, and combinations of blocks in the flowchart illustration, can be implemented by special purpose hardware-based systems which perform the specified actions or steps, or combinations of special purpose hardware and computer instructions.

Process **300** begins, after a start block, at block **302** where a request to save call information is received. In one embodiment of the present invention, call information is obtained from a call log, such as the call log shown in FIG. **4**. However, call information may also be obtained from other sources, such as dialer **210** or call validator **204** shown in FIG. **1**, without departing from the scope or spirit of the present invention.

In one embodiment of the present invention, when the user selects an entry in the call log, a context menu is displayed to the user. The context menu is configured to provide a user selectable choice to create a new contact, or to update an existing contact in the contact database. Briefly, FIG. **5** is an illustrative screen shot of an embodiment of a user-interface (UI) for a context menu **502** for updating or adding a contact to the contact database, in accordance with the present invention. When the user-interface includes a touch-sensitive display, a user may select an entry in the call log by tapping, touching, or otherwise triggering a selection within the call entry display field (such as tapping within call entry display field **402** of FIG. **4**).

In another embodiment, context menu **502** includes a view contact option (not shown) that a user may select to view an existing contact card. Moreover, the view contact option also is accessible from the dialer main screen.

In yet another embodiment of the present invention, a request to save call information may be initiated through a user selection of a SAVE command, such as during a current incoming or outgoing call.

Upon receiving a request to save call information, process control flow proceeds to decision block **304**.

At decision block **304**, a determination is made whether the call information is for a new contact or an existing contact in the contact database. In one embodiment of the present invention, the user selects from the context menu either to create a new contact card or to update an existing contact card. In one embodiment, when the user-interface includes a touch-sensitive display, selection within context menu **502** of FIG. **5** includes tapping, touching, or otherwise triggering the desired selection. In another embodiment of the invention, when the user selects to SAVE the call information, the user is provided the choice to create a new contact, or to update an existing contact in the contact database.

If it is determined at decision block **304**, that the user has selected to create a new contact in the contact database, process control flow proceeds to block **306**, where a new contact card is created employing the call information.

At block **306**, in one embodiment of the present invention, a new contact card is displayed to the user with the call information pre-populated into a data field. For example, the phone number within the call information may be pre-populated into a work telephone number data field, thus reducing the overall workload to the user. In another embodiment, the pre-populated data field may be at least one of a work, a home, a mobile, a pager, a car, a radio, and an assistant telephone number data field.

For example, FIG. **6** illustrates a screen shot of an embodiment of a UI for contact card **600** for a new contact with pre-populated phone number entry **602**, in accordance with the present invention.

In still another embodiment, the user may be provided with a user selectable menu for placing the phone number into a work, a home, a mobile, or a similar telephone data field, thereby increasing the flexibility to the user.

Upon completion of block **306**, process control flow moves to block **308**. At block **308**, additional contact information associated with the contact is received. Such contact information may include additional information the user wishes to save in the contact database, such as the contact's name and address. The user may also relocate pre-populated call information into a different data field. Process control flow continues to block **310**.

Back at decision block **304**, if it is determined that the user has selected to update an existing contact in the contact database, process flow control proceeds to block **312**.

At block **312**, a contact list of contact names in the contact database is provided to the user. FIG. **7** is an illustrative screen shot of an embodiment of a UI for contact list **700** from the contact database, in accordance with the present invention. Typically, contact list **700** is displayed as an alphabetically sorted list by contact's last name; however, the invention is not so limited. For example, the contact list may be sorted by the most recent call (incoming or outgoing) to the least recent call, or any other order, without departing from the scope or spirit of the present invention. Process flow control continues to block **314**.

At block **314**, a selection of the contact card to be updated is received from the user. In one embodiment of the present invention, an update contact card is provided to the user. FIG. **8** is an illustrative example of an embodiment of update contact card **800**, in accordance with the present invention. As shown in FIG. **8**, update contact card **800** is configured to enable the user to select a data field to update with the call information.

In another embodiment of the present invention, the user is provided with a predetermined user selectable menu for placing the phone number into a work, home, or mobile telephone number data field.

Upon completion of block **314**, process control flow moves to block **316**. At block **316**, the selected contact card is updated with the call information. For example, selection of the data field transfers the call information into that data field. Furthermore, the user may delete content of a data field, move content to a different data field, and update additional data fields for the selected contact card. For example, the user may select to insert the phone number into multiple telephone number data fields. Process control flow moves to block **310**.

At block **310**, the modified contact card is added to the contact database. If the contact card is an updated contact card, the previous contact card is deleted and replaced by the updated contact card. Upon completion of block **310**, the logical process control flow ends.

Although the above description has illustrated updating the contact database from a call log, the present invention is not so limited. For example, the contact database may also be updated from a telephone number obtained from a dialer, or a call in progress.

CONCLUSION

The above specification, examples, and data provide a complete description of the manufacture and use of certain embodiments of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

We claim:

1. A computer-readable medium having computer-executable instructions for updating a contact database in a mobile communications device, the instructions comprising:

receiving a request to save call information related to a phone call;

determining if the request to save the call information is an update to existing information in a contact card stored in the contact database or a request to create a new contact card in the contact database;

if the request is to update existing information,

retrieving a contact list of contact cards stored in the contact database;

receiving a selection of a contact card to be updated within the contact list;

updating the selected contact card with the call information related to the phone call;

replacing the existing contact card in the contact database with the updated contact card;

else if the request is to create a new contact card,

pre-populating a data field of the new contact card with call information;

receiving contact data to be associated with the new contact card;

modifying a data field in the new contact card with the received contact data; and

updating the contact database with die modified contact card.

2. The computer-readable medium of claim **1**, wherein the contact list comprises a list of at least one contact name associated with a contact card stored in the contact database.

3. The computer-readable medium of claim **1**, wherein the call information comprises a phone number.

4. The computer-readable medium of claim **1**, wherein the call information comprises a name of a caller.

5. The computer-readable medium of claim **1**, wherein the call information comprises a date of the phone call.

6. The computer-readable medium of claim **1**, wherein the call information comprises a time of the phone call.

7. The computer-readable medium of claim **1**, wherein the call information comprises a duration of the phone call.

8. The computer-readable medium of claim **1**, wherein the pre-populated data field includes at least one of a home phone number, a work phone number, and a mobile phone number.

9. The computer-readable medium of claim **1**, wherein the pre-populated data field is determined by evaluating a time associated with the phone call.

10. In a computer device having a graphical user interface and a user selection interface mechanism, a method of activating a selection for changing a contact database, comprising the steps of:

displaying a list of call entries in a call log of phone calls;

displaying a context menu in response to a user selection of an entry in the call log; and

displaying a plurality of options in the context menu, one of the options being to update a contact card in the contact database with call information from the selected entry in the call log, wherein an existing contact card is replaced with the updated contact card.

* * * * *

US007644376B2

(12) **United States Patent**
Karachale et al.

(10) **Patent No.: US 7,644,376 B2**
(45) **Date of Patent: Jan. 5, 2010**

(54) **FLEXIBLE ARCHITECTURE FOR NOTIFYING APPLICATIONS OF STATE CHANGES**

(75) Inventors: **Jan Karachale**, Sammamish, WA (US); **Jason William Fuller**, Bellevue, WA (US); **Robert Levy**, Virginia Beach, VA (US); **Zeke Koch**, Seattle, WA (US); **Ardan Arac**, Seattle, WA (US); **Brian Cross**, Redmond, WA (US); **Ori M. Amiga**, Seattle, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 411 days.

(21) Appl. No.: **10/873,881**

(22) Filed: **Jun. 22, 2004**

(65) **Prior Publication Data**

US 2005/0091219 A1 Apr. 28, 2005

Related U.S. Application Data

(60) Provisional application No. 60/513,723, filed on Oct. 23, 2003.

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **715/864**; 715/736; 715/859

(58) **Field of Classification Search** 715/736, 715/859; 455/412.2
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,098,093 A * 8/2000 Bayeh et al. 709/203
6,208,996 B1 * 3/2001 Ben-Shachar et al. 707/104.1
6,928,300 B1 * 8/2005 Skinner et al. 455/556.2
2002/0046299 A1 * 4/2002 Lefeber et al. 709/318
2003/0028602 A1 2/2003 Bhattacharya 709/206

OTHER PUBLICATIONS

S. Ethier, "Application-Driven Power Management: A Framework for Achieving Fine-Grained Control Over the Power Consumption of Purpose-Specific Mobile Devices", QNX Software Systems Ltd., pp. 3-15.

J. Inouye et al., "Dynamic Network Reconfiguration Support for Mobile Computers", MOBICOM 97, pp. 13-22, 1997.

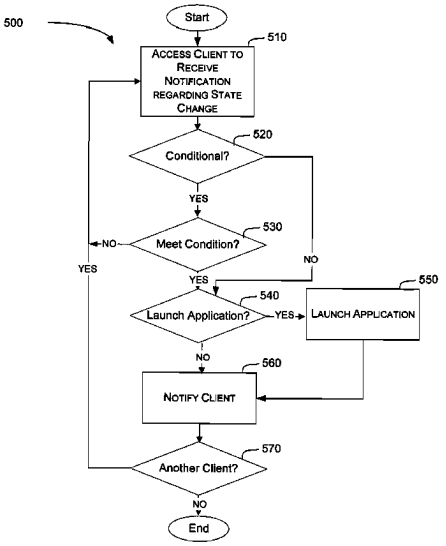
* cited by examiner

Primary Examiner—Weilun Lo
Assistant Examiner—Kim-Lynn Dam
(74) *Attorney, Agent, or Firm*—Merchant & Gould, P.C.

(57) **ABSTRACT**

Described is a method and system a unified mechanism for storing device, application, and service state, as well as a rich notification brokerage architecture. Clients register with a notification broker to receive notifications for changes to state properties. When a registered state property changes, a notification broker determines which clients to notify of the state change and provides the client with a notification regarding the change. Clients may be notified whenever a state changes, when a state change meets a predetermined condition, or based on a schedule. An application may also be launched in response to a state change. An application programming interface (API) is provided that provides a unified way of accessing state change information across different components within the device.

25 Claims, 5 Drawing Sheets



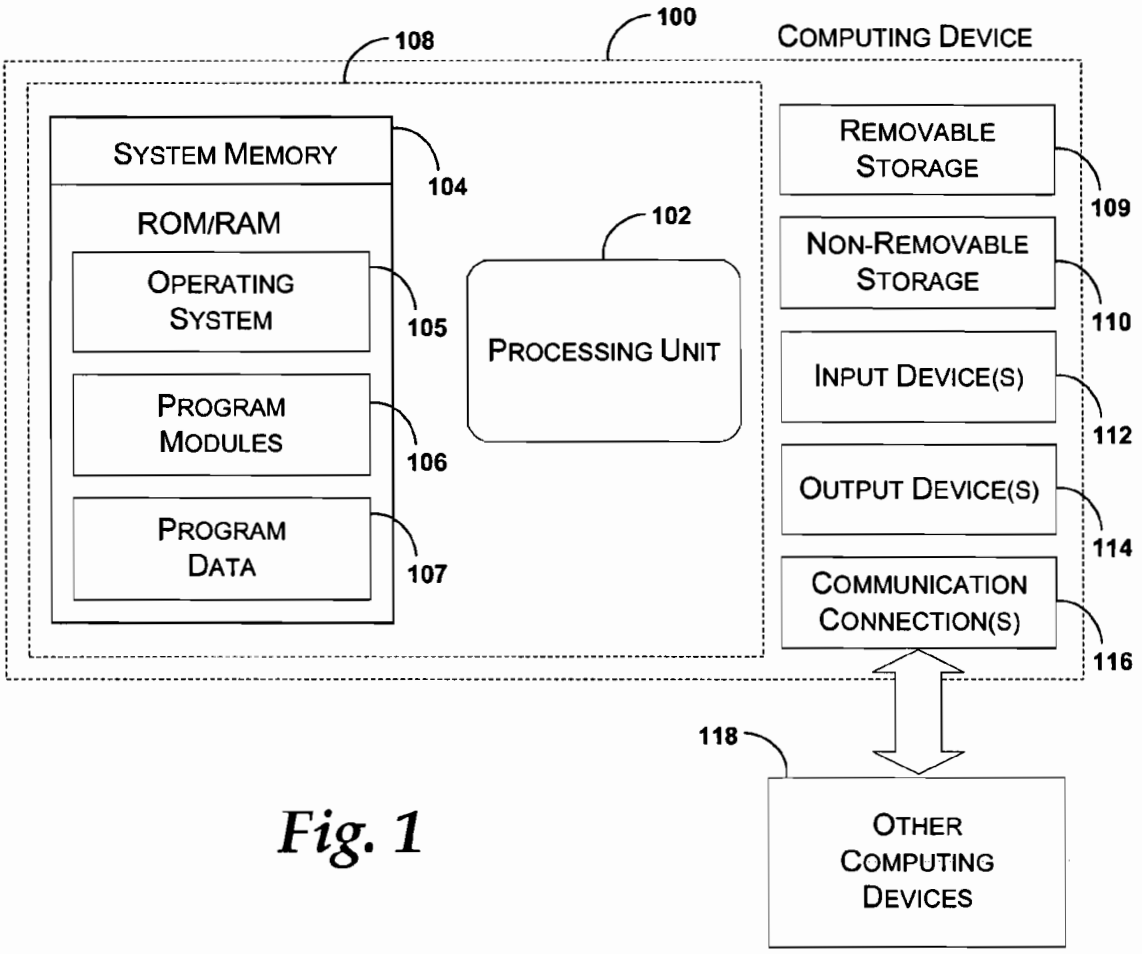

100
COMPUTING DEVICE
108
SYSTEM MEMORY
104
ROM/RAM
OPERATING SYSTEM
105
PROGRAM MODULES
106
PROGRAM DATA
107
102
PROCESSING UNIT
REMOVABLE STORAGE
109
NON-REMOVABLE STORAGE
110
INPUT DEVICE(S)
112
OUTPUT DEVICE(S)
114
COMMUNICATION CONNECTION(S)
116
118
OTHER COMPUTING DEVICES

Fig. 1

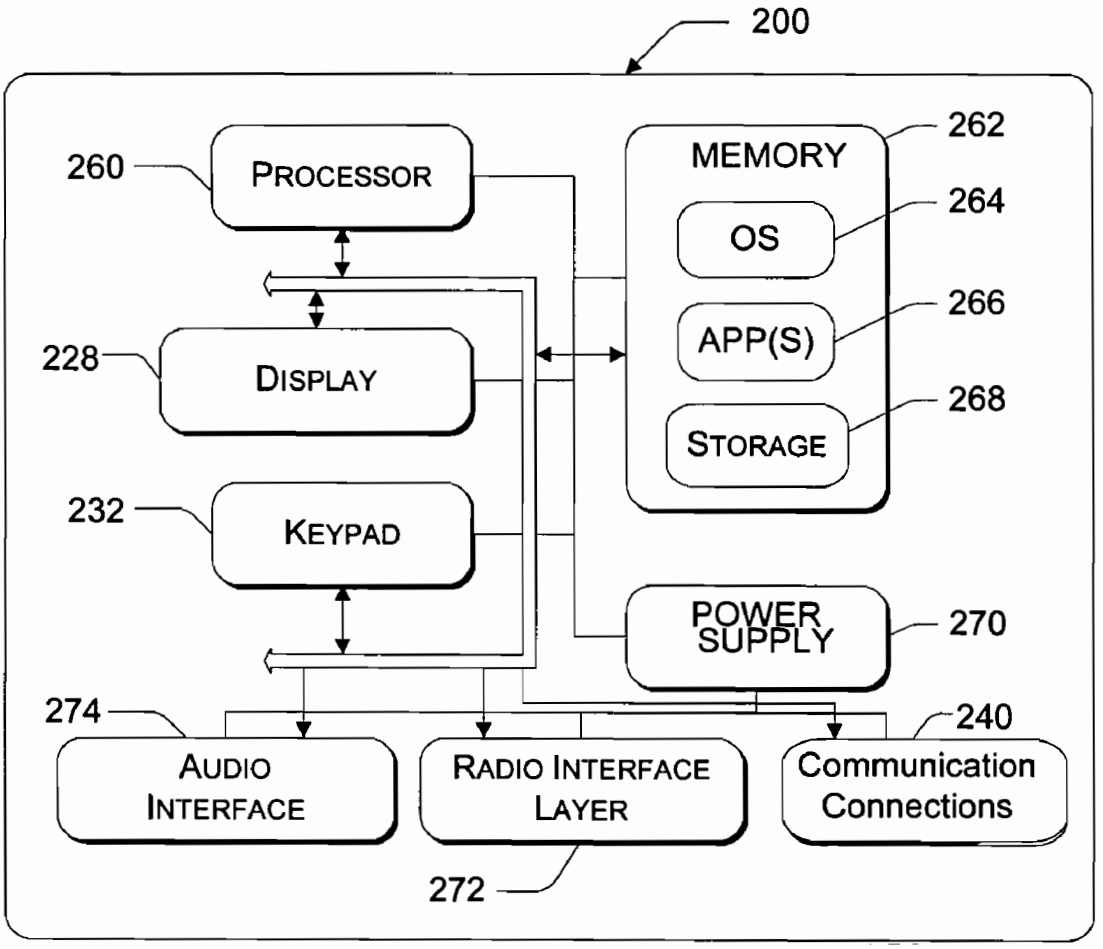

200
260
PROCESSOR
MEMORY
262
264
OS
266
APP(S)
228
DISPLAY
268
STORAGE
232
KEYPAD
POWER SUPPLY
270
274
240
AUDIO INTERFACE
RADIO INTERFACE LAYER
Communication Connections
272

Fig. 2

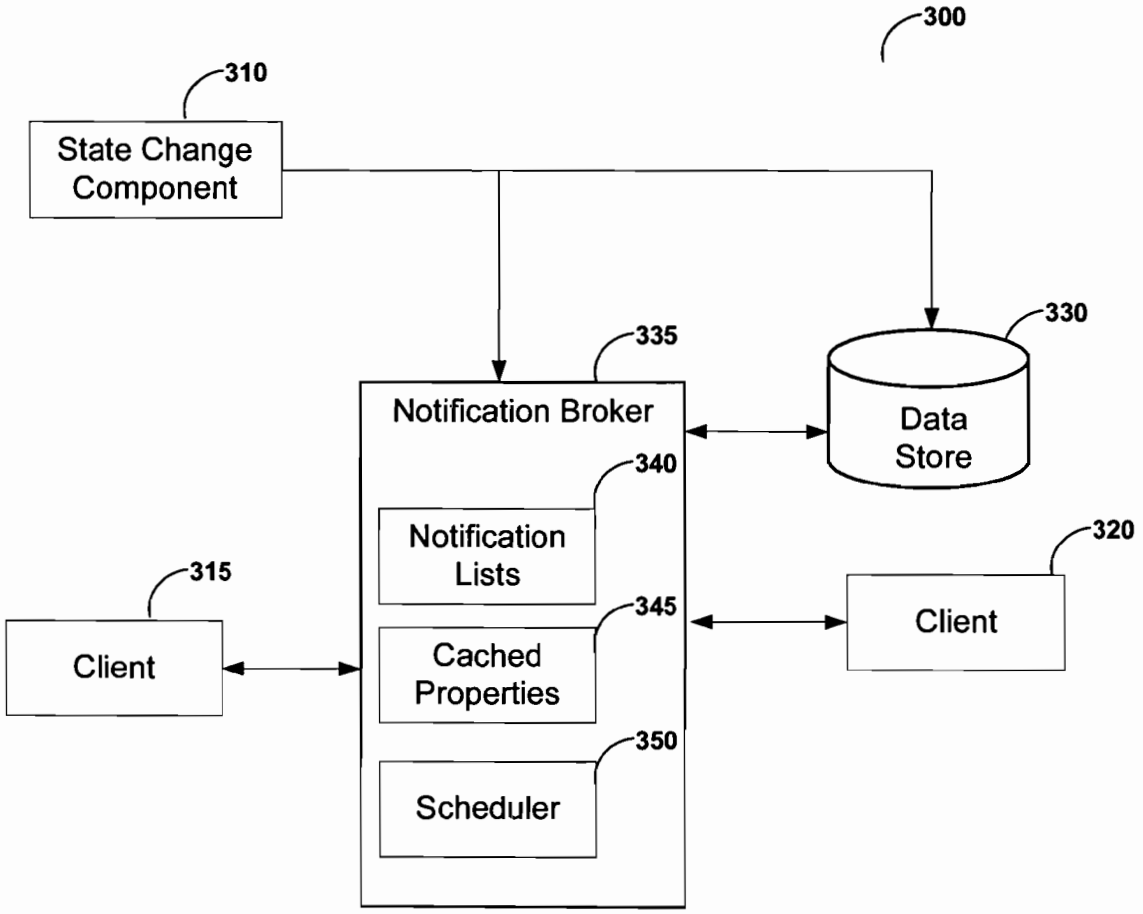

300
310
State Change Component
330
335
Notification Broker
Data Store
340
Notification Lists
320
315
Client
345
Client
Cached Properties
350
Scheduler

Fig. 3

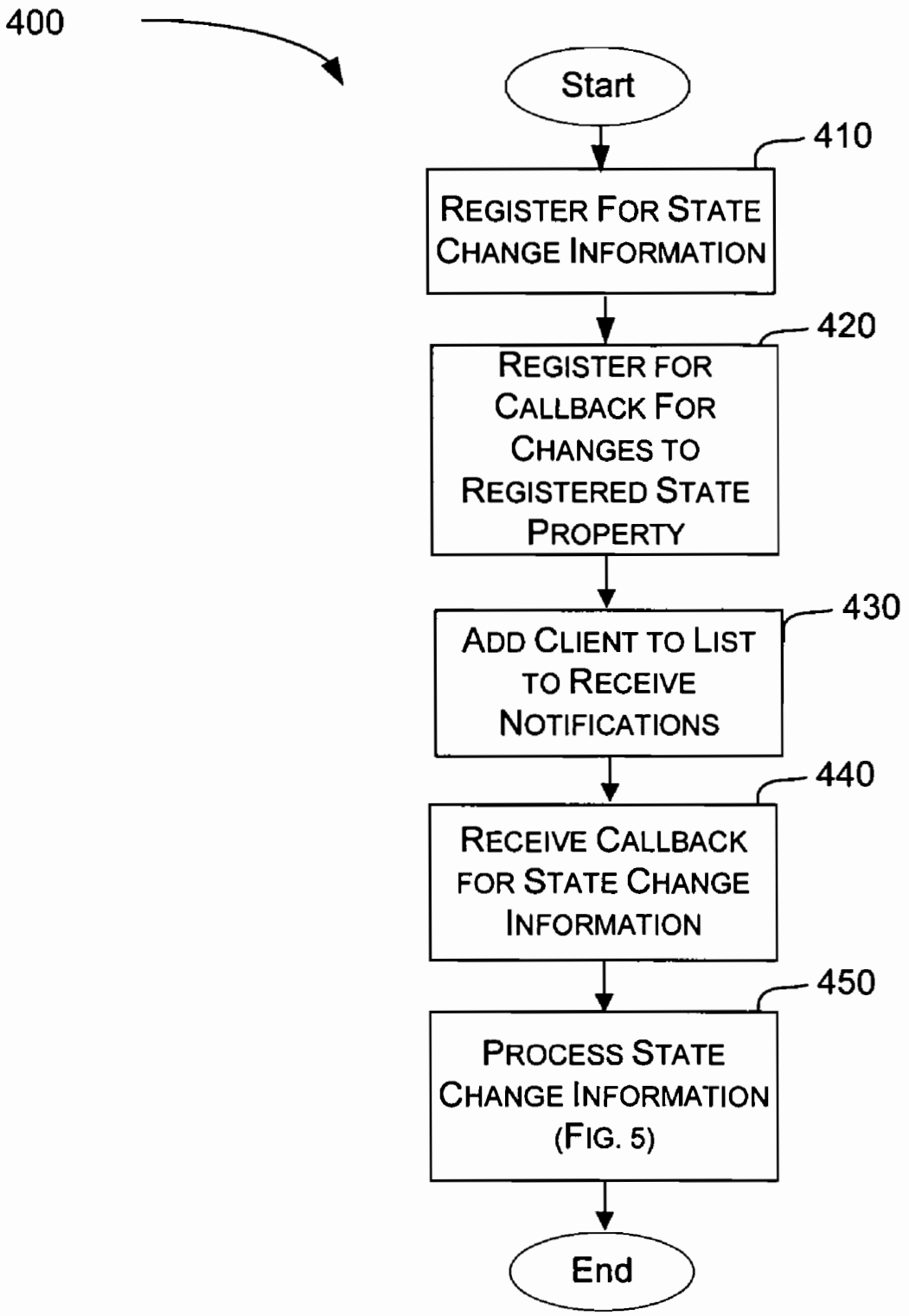

400
Start
410
REGISTER FOR STATE CHANGE INFORMATION
420
REGISTER FOR CALLBACK FOR CHANGES TO REGISTERED STATE PROPERTY
430
ADD CLIENT TO LIST TO RECEIVE NOTIFICATIONS
440
RECEIVE CALLBACK FOR STATE CHANGE INFORMATION
450
PROCESS STATE CHANGE INFORMATION (FIG. 5)
End

Fig. 4

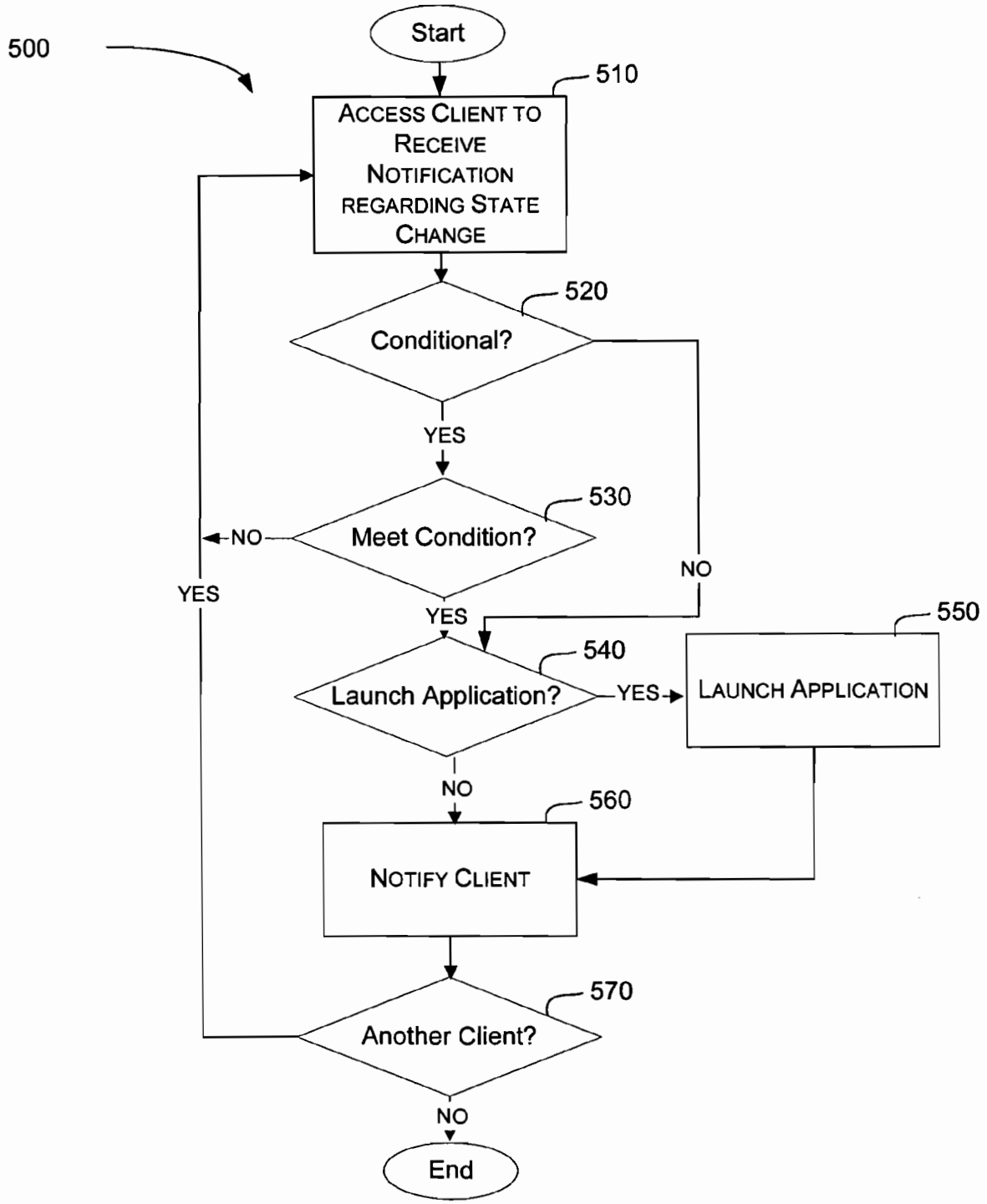

500
Start
510
ACCESS CLIENT TO RECEIVE NOTIFICATION REGARDING STATE CHANGE
520
Conditional?
YES
NO
530
Meet Condition?
NO
YES
YES
NO
550
540
Launch Application?
YES
LAUNCH APPLICATION
NO
560
NOTIFY CLIENT
570
Another Client?
NO
End

Fig. 5

FLEXIBLE ARCHITECTURE FOR NOTIFYING APPLICATIONS OF STATE CHANGES

RELATED APPLICATIONS

This utility patent application claims the benefit under 35 United States Code §119(e) of U.S. Provisional Patent Application No. 60/513,723 filed on Oct. 23, 2003.

BACKGROUND OF THE INVENTION

Today, mobile devices are designed to run a variety of applications and keep a user updated with current information. Some of these devices include personal digital assistants, wireless phones, and email devices. Mobile devices are now capable of connecting to the Internet and other networks thorough various means and thus exchange information over the networks. These mobile devices may update applications and send and receive information, such as emails, attachments to emails, and web page content. Providing all of this functionality requires applications on the mobile device to be notified of various events, such as when a new email is available, when a screen of the device is activated, when a phone call is received, and the like. It is difficult, however, to access all of the different state changes associated with the device.

SUMMARY OF THE INVENTION

Briefly described, the present invention is directed at unifying state and notification architecture across devices.

According to one aspect of the invention, clients register with a notification broker to receive notifications for changes to state properties. When a registered state property changes, a notification broker determines which clients to notify of the state change and provides the client with a notification regarding the change. For example, a client may register to receive notifications regarding changes to battery strength, network connectivity, memory usage, and the like. Whenever one of these registered state properties changes, the notification broker sends the client a notification message.

According to another aspect of the invention, clients may be notified whenever a state changes, when a state change meets a predetermined condition, or based on a schedule.

According to yet another aspect of the invention, an application may be launched in response to a state change or a schedule. For example, a client may register to have an application started when a certain event occurs, such as the mobile device receiving a message directed toward the application to be launched. The application may also be started based on a schedule configured by the client.

According to yet another aspect of the invention, an application programming interface (API) is provided that is directed to providing a unified way of accessing state change information across different components within the device. For example, an application may use the same function call to access state properties set by different components within the device.

According to still yet another aspect of the invention, the registered state properties may persist across device reboots.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates an exemplary computing device;

FIG. **2** shows an exemplary mobile device;

FIG. **3** illustrates an exemplary state management and notification system;

FIG. **4** illustrates a process for a state change notification system; and

FIG. **5** shows a process for processing state change information, in accordance with aspects of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Briefly described, the present invention is directed to providing a method and system a unified mechanism for storing device, application, and service state, as well as a rich notification brokerage architecture. Generally, clients register with a notification broker to receive notifications when certain state properties change. When a registered state property changes, the notification broker determines which clients to notify of the state change and provides the client with a notification regarding the change. Clients may be notified whenever a state changes, when a state change meets a predetermined condition, or based on a schedule. An application may also be launched in response to a state change or a schedule. An application programming interface (API) is also provided that is directed at providing a unified way of accessing state change information across different components within the device.

Throughout the specification and claims, the following terms take at least the meanings explicitly associated herein, unless the context clearly dictates otherwise. The meanings identified below are not intended to limit the terms, but merely provide illustrative examples for the terms. The meaning of "a," "an," and "the" includes plural reference, the meaning of "in" includes "in" and "on."

The term "state property" refers to a "status" variable registered and stored with the notification system for maintenance and change-notifications.

The term "notification request" refers to a request from a client to be notified of a state change.

The term "notification list" refers to a collection of clients which have registered for state property change notifications.

The term "notification broker" refers to an underlying driver responsible for adding, updating, and removing data from a data store.

The term "state change component" refers to any component which adds, updates, or generally maintains State Properties in the data store.

The term "client" refers to any component which registers for state property change notifications. A client may be a state change component as well as a state change component being a client.

The term "state property identifier" refers to a "friendly" string (name) representation of the State Property. This identifier may be hierarchical and is unique.

The term "conditional notification" refers to a notification that is sent when a state property changes and the new value of the state property meets the condition that was specified in the notification request.

Illustrative Operating Environment

With reference to FIG. **1**, an exemplary system for implementing the invention includes a computing device, such as computing device **100**. Computing device **100** may be configured as a client or a server. In a very basic configuration, computing device **100** typically includes at least one processing unit **102** and system memory **104**. Depending on the exact configuration and type of computing device, system memory **104** may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. System memory **104** typically includes an operating system

105, one or more program modules 106, and may include program data 107. This basic configuration is illustrated in FIG. 1 by those components within dashed line 108.

Computing device 100 may have additional features or functionality. For example, computing device 100 may also include additional data storage devices (removable and/or non-removable) such as, for example, magnetic disks, optical disks, or tape. Such additional storage is illustrated in FIG. 1 by removable storage 109 and non-removable storage 110. Computer storage media may include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. System memory 104, removable storage 109 and non-removable storage 110 are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computing device 100. Any such computer storage media may be part of device 100. Computing device 100 may also have input device(s) 112 such as keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 114 such as a display, speakers, printer, etc. may also be included.

Computing device 100 also contains communication connections 116 that allow the device to communicate with other computing devices 118, such as over a network. Communication connections 116 are one example of communication media. Communication media may typically be embodied by computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as a carrier wave or other transport mechanism, and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. The term computer readable media as used herein includes both storage media and communication media.

With reference to FIG. 2, one exemplary system for implementing the invention includes a mobile device, such as mobile device 200. The mobile device 200 has a processor 260, a memory 262, a display 228, and a keypad 232. The memory 262 generally includes both volatile memory (e.g., RAM) and non-volatile memory (e.g., ROM, Flash Memory, or the like). The mobile device 200 includes an operating system 264, such as the Windows CE operating system from Microsoft Corporation or other operating system, which is resident in the memory 262 and executes on the processor 260. The keypad 232 may be a push button numeric dialing pad (such as on a typical telephone), a multi-key keyboard (such as a conventional keyboard). The display 228 may be a liquid crystal display, or any other type of display commonly used in mobile devices. The display 228 may be touch-sensitive, and would then also act as an input device.

One or more application programs 266 are loaded into memory 262 and run on the operating system 264. Examples of application programs include phone dialer programs, email programs, scheduling programs, PIM (personal information management) programs, word processing programs, spreadsheet programs, Internet browser programs, and so forth. Application programs 266 may use a common API to perform actions on state properties associated with the device. For example, a phone dialer program may register with a notification system to receive notifications regarding changes to signal strength, phone state, battery strength, and the like. The mobile computing device 200 also includes non-volatile storage 268 within the memory 262. The non-volatile storage 268 may be used to store persistent information which should not be lost if the mobile computing device 200 is powered down. The applications 266 may use and store information in the storage 268, such as e-mail or other messages used by an e-mail application, contact information used by a PIM, appointment information used by a scheduling program, documents used by a word processing program, and the like.

The mobile computing device 200 has a power supply 270, which may be implemented as one or more batteries. The power supply 270 might further include an external power source, such as an AC adapter or a powered docking cradle that supplements or recharges the batteries.

The mobile computing device 200 may also include external notification mechanisms, such as an LED (not shown) and an audio interface 274. These devices may be directly coupled to the power supply 270 so that when activated, they remain on for a duration dictated by the notification mechanism even though the processor 260 and other components might shut down to conserve battery power. The audio interface 274 is used to provide audible signals to and receive audible signals from the user. For example, the audio interface 274 may be coupled to a speaker for providing audible output and to a microphone for receiving audible input, such as to facilitate a telephone conversation.

Mobile computing device 200 may also contain communication connections 240 that allow the device to communicate with other computing devices, such as over a wireless network. The mobile computing device 200 also includes a radio interface layer 272 that performs the function of transmitting and receiving radio frequency communications. The radio interface layer 272 facilitates wireless connectivity between the mobile computing device 200 and the outside world, via a communications carrier or service provider. Transmissions to and from the radio interface layer 272 are conducted under control of the operating system 264. In other words, communications received by the radio interface layer 272 and communication connections 240 may be disseminated to application programs 266 via the operating system 264, and vice versa.

Illustrative State Change Notification System

FIG. 3 illustrates an exemplary state management and notification system, in accordance with aspects of the invention. Notification system 300 includes state change component 310, clients 315 and 320, notification broker 335, and data store 330. According to one embodiment, notification broker 335 also maintains notification lists 340, cached properties 345 and scheduler 350.

Clients, such as client 315 or 320, register to receive notifications regarding changes to a state property with notification broker 335. Generally, a client may make a notification request with notification broker 335 that registers the client to receive notifications whenever a state property changes, when the change meets a conditional, or based upon a schedule. The notifications may be permanent or transient notifications.

Permanent notifications are kept in a data store (320). According to one embodiment, the permanent notifications are maintained in a back-end data store, such as the registry, and hence are "persisted" across reboots. As these notifications are persisted, these types of state properties have the

same value before a soft-reset (or shutdown) as they do upon a restart (or boot). According to one embodiment, state properties are persistent by default.

Transient notifications are not permanent and are, therefore, not persisted across reboots. In other words, if a device is soft-reset (or rebooted) the notification request is deleted from notification list **340**. In order to restore a transient notification, a client re-registers (sends another notification request to broker **335**) to receive notifications regarding changes to the state property.

A client may also register to have an application launched upon the occurrence of a state change and/or according to a schedule. Generally, notification broker **335** issues a command to start an application specified by the client if the application is not already running when the state change or the scheduled time occurs. According to one embodiment, the client can specify command-line parameters to be passed into the application when it is launched. If the launched process creates a window then a message is sent to the window indicating the notification. If the target process is already running on the client, then the client simply receives the notification message.

Notifications sent to clients may also be batched. Batched state properties are intended for use by state properties which may undergo frequent value changes. A predetermined period of time is set that allows the state property value to "stabilize." According to one embodiment, the predetermined period is set to 200 ms. If no changes are made to the state property value during the predetermined period, the notification is delivered to the registered clients. The batching predetermined period is configurable and is stored in data store **330**.

Data store **330** is configured to store registered state properties, as well as other information. According to one embodiment, data store **330** is the registry provided with an operating system, such as the registry provided with the Windows XP operating system provided by Microsoft Corporation. Data store **330** may also be any other type of data store in which information may be set and accessed. Data store **330** may also comprise one or more data stores maintained at various locations within notification system **300**.

Data store **330** may also be pre-loaded with a default set of state property data that may be accessed by clients **315** and **320**. Pre-loading the state property data makes many of state properties available without the client having to add a state property. For example, according to one embodiment, the following states are available to clients without registering the state: Display Orientation (Resolution, Brightness); Undismissed reminders (Count, Subject, Date, Time; Location); Undismissed alarms (Count, Description, Date, Time); Battery (% remaining, State); Backup battery (% remaining, State); Memory (Program memory free, Program memory used, Storage memory free, Storage memory used); Storage card (Total memory free, Total memory used); Hardware (Flip-phone state (open/closed), Keyboard enabled, Wifi enabled, Bluetooth enabled, Headphones present, Camera present); Messaging (Unread count, Total count, Drafts count, Outbox count); Tasks (High priority count, Due today count, Overdue count); Calendar (Next appointment, Name, Location, Date, Time, POOM ID); All day appointment (Name, Location, Date, Time, POOM ID); Current appointment (Name, Location, Date Time, POOM ID); Current free/busy status; Instant Messenger (Status, Online contacts count; Offline contacts count); Connectivity (Speed, Wifi, Access point, Signal strength, Connection name, Connects to (work, internet), Status); Media player (Status, Playlist (Name, Mode (repeat, shuffle), Track count, Total duration), Track (Type (audio, video), Name, Album, Artists, Track #, Genre, Year, Duration, Play position, Filename, Bit rate)); Sync status; Telephony (Operator, Signal strength, Phone state, Profile, Roaming, Radio state, Active call (Caller name, Caller number), Missed call count, SIM toolkit message. As can be seen, the states span across many different applications.

According to one embodiment, the state property data is organized into a hierarchy. The hierarchy allows a client to specify a group of state properties by referencing the parent. The state property may be set to a data type, such as string, integer, float, and the like.

The client may identify the state property by a "friendly" string (name) representation of the state property. For example, "battery\a" references the state property "a" under the parent "battery", and likewise there could be a "battery\b" which would be the state property "b" under the same parent. When referring to a group of state properties under a common parent, then the parent identifier may be used. For example to receive notifications based on changes to all of the battery states, then "battery" would be provided in the registration, thereby referencing all of the battery state properties using a single name.

Broker **335** may be configured to control access to setting/adjusting/removing state property types within data store **330**. For example, a restriction could be placed on a state property limiting the deletion of the property from the notification system to a predetermined list of clients. When a state property is deleted, clients that have registered for notifications relating to the property are notified of its deletion.

As discussed above, clients **315** and **320** register for the state properties they are interested in receiving notifications about when the state property changes. Clients may register to receive a notification whenever the state they registered changes, when a conditional applied to the state value meets a condition, or upon a predetermined schedule. A conditional notification may be based upon many different conditions. According to one embodiment, the conditionals include: all, equal, not equal, greater than, greater or equal than, less than or equal, less than, contains, starts with, and ends with. For example, client **315** may register with notification broker **335** to receive a notification when the missed call count state property is Greater than fifteen and when the caller name Contains "Ja." Conditionals allow a client to receive the state change information they are interested in without having to process state change information they do not care about.

The clients registered to receive notifications regarding changes to state properties are maintained in notification lists **340**. Notification broker **335** accesses notification lists **340** to determine the clients that should receive notifications when a registered state property has changed.

Scheduler **350** may be configured to notify and/or activate a client based on a schedule. The scheduled activation notification mode allows a client to receive a notification based on a simplified recurring schedule registered with notification broker **335**. Schedules may be configured to occur at any interval, such as on the scale of seconds, minutes, hours, days, weeks, or months. According to one embodiment, schedules are defined by the date/time of the first occurrence and the amount of time between occurrences. Additionally, schedules may be defined without a recurrence interval. When no recurrence interval is provided, the notification is only sent once and then the registration is removed from the notifications list. Additionally, when a notification arrives, if the specified application path (provided during the notification request) cannot be found, the scheduled notification registration is removed from the notification list **340**.

State change component **310** updates the value of the state property within data store **330** when the state changes. State change component **310** may update the state directly in data store **330** or through notification broker **335**. When the state is updated through data store **330**, data store **330** communicates the change to notification broker **335**. When the state is updated through notification broker **335** then notification broker **335** updates the state in data store **330**. In either case, notification broker **335** determines which clients should be notified based on the state change. Notification broker **335** parses notification lists **340** and determines the clients that have registered for notifications regarding the state change. Notification broker **335** applies any conditionals to the value of the state property that has changed and notifies the client when the conditional is met. When there is not a conditional associated with the state change, the client is notified of the state change.

When a client, such as client **315** and client **320**, receives a notification from notification broker **335**, the client may call a function within a common API (see discussion below) to retrieve the new state of the state property. Alternatively, the property information may be directly delivered to the client along with the notification. When the client is no longer interested in receiving notifications relating to a particular state property, the client may un-register itself from receiving change notifications relating to the state property. Clients **315** and **320** may also directly query broker **335** at any time to find the state of a state property using the common API.

State property values may also be cached by notification broker **335** in cached properties **345**. A state property value may be cached for many different reasons. For example, a state property value may be cached such that a previous value of the state property may be compared with a current value of the state property. Additionally, the cache may help to facilitate quick repeated lookups requesting the value of the state property.

According to one embodiment, notification system **300** supports NET (managed) clients for both additions to the store, as well as change notification registrations.

The following are some exemplary scenarios to further clarify state management notification system **300**.

EXAMPLE 1

ISV Services

Norm the Newbie has built a C# application which keeps a complete database of the current season's Baseball statistics (e.g., teams, players, scores, stats, etc.). He has also built a simple XML web-services client which can connect to a sports website and pull-down updated daily statistics. Since the amount of data the application stores is relatively large, Norm only wants his application to sync data when a "fat pipe" (e.g., 802.1x) is available on the device (e.g., PPC). Norm then registers his application by sending a notification request to notification broker **335** for notifications when a high-bandwidth connection is available. Norm additionally specifies in the notification request to launch his application when the high-bandwidth connection is available. When the state change component associated with the connection updates the state of the connection, notification broker **335** activates Norm's app when the state indicates that it is a high-speed connection.

EXAMPLE 2

Corporate LOB (Line of Business) Applications

Elph the Enterprise developer has built a field-service form-based VB.Net application for insurance adjuster usage. The application allows an insurance adjuster to look-up part #s and costs, make notes, retrieve car schematics, and the like. Each day, the insurance adjuster takes his mobile computing device out in the field to service customers. The application persists all of its data for today's operation locally in a data store. Elph would like the application to synchronize the offline store with the company's main servers each time the device is cradled. Therefore, Elph registers his application for notifications on synchronization cradle events. Whenever the device is cradled, the application is notified and the application synchronizes its data.

EXAMPLE 3

Phone Game

Golem the phone game developer is building a next-generation multi-player RPG for a phone. He anticipates the game will be so very popular that it will last for weeks and months at a time. One of Golem's key concerns is the persistence of game state without user intervention. One of the game's neat features is the ability to save current state right before the phone runs out of batteries and ensure the user never loses any data. Golem registers his application to receive low battery notifications to ensure that the game information will be saved before the device runs out of batteries.

EXAMPLE 4

Device Management Client

Eric the emerging Enterprise Management Client developer is looking to create the next generation mobile computing device and phone management client; able to handle client updates, virus scanning, policy enforcement, and more. Using C# he has built a power device-side client which can handle requests based on server-driven commands. Each night at 3 am, Eric would like his application "woken up" so he can contact the server for updated policies, virus scanner signatures, and the like. In order to achieve this, he simply registers his application with notification broker **335** for a scheduled notification (each day at 3 am). Eric is now assured his app will run at the specified time.

FIG. **4** illustrates a process for a state change notification system, in accordance with aspects of the invention. After a start block, process **400** flows to block **410**, where a client registers to be notified of changes to at least one state property. If the state property is not already being monitored by another client, the state property is added to the list of available state properties. As discussed above, a list of available properties is pre-loaded into the notification system. The client may register to receive notification on all changes made to the property, changes that meet a condition, as well being notified according to a schedule.

Moving to block **420**, a callback is registered with the notification system such that when a change is made to a registered state property, the notification system is made

aware of the change. According to one embodiment, a notification broker registers a callback with the operating system registry for changes made to the state property value.

Flowing to block, **430**, the client is added to a notification list to receive notification messages relating to the state property. Clients included in the notification list receive notifications regarding changes to the registered state property.

Transitioning to block **440**, a callback is received when a change is made to any of the registered state properties. According to one embodiment, the callback includes an identifier identifying the state property changes, as well as the current value of the state property.

Moving to block **450**, the state change information is processed. Generally, processing the state change information includes determining if any conditionals, schedules, batches, or application launching conditions, apply to each of the registered clients for the changed state property (See FIG. **5** and related discussion).

FIG. **5** shows a process for processing state change information, in accordance with aspects of the invention. After a start block, process **500** flows to block **510** where a client registered for receiving notifications regarding a state change for the changed state property is accessed. According to one embodiment, a notification list is accessed to determine the registered clients for the state property that has changed.

Moving to decision block **520**, a determination is made as to whether the client has specified any conditionals that are to be applied to the state property before notifying the client.

When a conditional expression is associated with the notification request, the process flows to decision block **530** where a determination is made as to whether the condition is met. When the condition is met, or when the client has not specified any conditionals, the process moves to decision block **540**, where a determination is made as to whether the client has specified to launch an application in response to a change to the state property. When the client has specified to launch the application, the process moves to block **550** where the application is launched if it is not already running. When the client has not specified to launch the application, or the application has been launched at block **550**, the process moves to block **560** where the client is notified of the state change.

The process then flows to decision block **570**, where a determination is made as to whether there are more clients registered to receive a notification regarding a change to the state property. When there is another client, the process returns to block **510**. When there are no other clients, the process then moves to an end block and returns to processing other actions.

State Property Types and Modes

According to one embodiment of the invention, two APIs may be used to access the state information in the notification system. A native, or underlying API is provided and a managed API is provided to clients. The managed API accesses the native API to perform its operations.

The following is an exemplary native API, in accordance with aspects of the invention:

```
#define E_ALREADY_REGISTERED      ERROR_ALREADY_REGISTERED
#define E_DATATYPE_MISMATCH       ERROR_DATATYPE_MISMATCH
#define E_INSUFFICIENT_BUFFER     ERROR_INSUFFICIENT_BUFFER
#define E_INVALID_HANDLE          ERROR_INVALID_HANDLE
#define E_NOT_FOUND               ERROR_NOT_FOUND
#define E_NOT_READY               ERROR_NOT_READY

DECLARE_HANDLE(HREGNOTIFY); // transient notification handle

// ********************************************************
// Enumeration Name: REG_COMPARISONTYPE

// Purpose: used to define how state property values should be compared to
```

```
//      target values for conditional change notifications. When executing
//      the REG_COMPARISONTYPE, the changed value is used as the
//      l-value, that is, REG_CT_LESS would mean fire the notification if
//      the changed value is less than the target value.
//
// Description:
//  the following shows what statements will be true when the comparison is
//  done. A case insensitive CompareString is
//  used to compare the strings, the following information is used to clarify
//  the intent of the REG_COMPARISONTYPE values.
//  Let "cv" represent the changed value and "psz" or "dw" represent the
//  TargetValue specified in the NOTIFICATIONCONDITION structure. Let "l"
//  represent the length of the string specified in psz (wcslen(psz)) and
//  "cvl" represents the string length of the changed value (wcslen(cv)).
//                                  REG_SZ              REG_DWORD
//  REG_CT_EQUAL                    | (0 == strcmp(cv, psz))      | (cv == dw) |
//  REG_CT_NOT_EQUAL                | (0 != strcmp(cv, psz))      | (cv != dw) |
//  REG_CT_GREATER                  | (0 <  strcmp(cv, psz))      | (cv >  dw) |
//  REG_CT_GREATER_OR_EQUAL         | (0 <= strcmp(cv, psz))      | (cv >= dw) |
//  REG_CT_LESS                     | (0 >  strcmp(cv, psz))      | (cv <  dw) |
//  REG_CT_LESS_OR_EQUAL            | (0 >= strcmp(cv, psz))      | (cv <= dw) |
//  REG_CT_CONTAINS                 | (0 != strstr(cv, psz))      |   N/A   |
//  REG_CT_STARTS_WITH              | (0 == strncmp(cv, psz, l))  |   N/A   |
//  REG_CT_ENDS_WITH                | (0 == strcmp(cv+cvl-l, psz))|   N/A   |
// ************************************************
typedef enum tagREG_COMPARISONTYPE
{
  REG_CT_EQUAL,
  REG_CT_NOT_EQUAL,
  REG_CT_GREATER,
```

```
  REG_CT_GREATER_OR_EQUAL,
  REG_CT_LESS,
  REG_CT_LESS_OR_EQUAL,
  REG_CT_CONTAINS,
  REG_CT_STARTS_WITH,
  REG_CT_ENDS_WITH
} REG_COMPARISONTYPE;

// ************************************************************
// Structure Name: NOTIFICATIONCONDITION
//
// Purpose: used to define a condition under which property state change
//       notifications should be fired
//
// Description:
//   REG_COMPARSIONTYPE ctComparisonType - how to compare the changed
value with the value specified in TargetValue
//   DWORD dwMask - If this value is not set to 0 then TargetValue is
//            interpreted as a DWORD. This mask is applied to the changed
//            DWORD value before the comparison is done, the mask is
//            not applied to TargetValue. If this value is set to 0 then
//            TargetValue is interpreted as a string.
//   union TargetValue - if dwMask is set to 0 then the comparison is done
//               between psz and the changed value. If dwMask is set
//               to any value other than 0 then the comparison is done
//               between dw and the masked (see dwMask) changed value.
//               If the changed value is not of type REG_SZ nor
//               REG_DWORD or if the changed value is of type REG_DWORD
//               and dwMask is set to 0 or if the changed value is of
//               type REG_SZ and dwMask is not set to 0, then the
```

```
//                notification is ignored. If the changed value is
//                deleted and the comparison is to be done against dw,
//                0 is used as the changed value and the comparison
//                proceeds as usual. If the value is deleted and the
//                comparison is to be done against psz, NULL is used as
//                the changed value and only REG_CT_EQUAL and
//                REG_CT_NOT_EQUAL are processed by the comparison, the
//                deletion notification is ignored if any other
//                ctComparisonType is used.
//
// ***************************************************
typedef struct tagNOTIFICATIONCONDITION
{
  REG_COMPARISONTYPE ctComparisonType;
  DWORD dwMask;
  union
  {
    LPCTSTR psz;
    DWORD dw;
  } TargetValue;
} NOTIFICATIONCONDITION;


// // Function Prototype: REGISTRYNOTIFYCALLBACK
//
// Purpose: defines the protoype of the callback used by RegisterNotifyCallback
//
// Arguments:
//   IN HREGNOTIFY hNotify - the handle to a valid HREGNOTIFY, this is the
//                same handle returned from RegistryNotifyCallback
```

```
//   IN DWORD dwUserData - user data that was passed to RegistryNotifyCallback
//   IN const PBYTE pData - a pointer to the new value for the value, this
//                  is set to NULL if the value was deleted
//   IN const UINT cbData - the number of bytes to pointed to by pData, this
//                  value will be set to 0 if the value was deleted
//
// Description:
//   this callback used to notify clients that registered for notifications
//   using RegistryNotifyCallback. It is safe to call RegistryCloseNotification
//   from within the callback if no further notification are required.
//
// ************************************************
typedef void (*REGISTRYNOTIFYCALLBACK)(HREGNOTIFY hNotify,
                        DWORD dwUserData,
                        const PBYTE pData,
                        const UINT cbData);

// ************************************************************
// Function Name: RegistryGetDWORD
//
// Purpose: used to read a REG_DWORD registry value
//
// Arguments:
//   IN HKEY hKey - handle to a currently open key, or a predefined root value
//   IN LPCTSTR pszSubKey - the key under which the value is stored (if this
//                  value is null pszValueName is assumed to be under
//                  hKey)
//   IN LPCTSTR pszValueName - the name of the value to retrieve (may be NULL
//                    to retrieve the default value)
//   OUT DWORD * pdwData - a pointer to the buffer which will receive the
```

```
//                  data associated with the value
//
// Return Values:
//   HRESULT
//   S_OK - the data was copied to the buffer
//   E_INVALIDARG - hKey or pdwData is invalid
//   E_DATATYPE_MISMATCH - the value is not of type REG_DWORD
//   An error value returned from RegOpenKeyEx or RegQueryValueEx wrapped as
//   a FACILITY_WIN32 HRESULT
//
// Results:
//   SUCCEEDED - pData points to the data associated with the value
//   FAILED - no change.
//
// Description:
//   the DWORD associated with the value is copied to the buffer pointed to
//   by pData. If the key pointed to by hKey+pszSubKey does not exist,
//   RegistryGetDWORD will fail since it uses RegOpenKey to access the key
//
// ***************************************************
HRESULT WINAPI RegistryGetDWORD(HKEY hKey,
                    LPCTSTR pszSubKey,
                    LPCTSTR pszValueName,
                    DWORD * pdwData);


// ***************************************************
// Function Name: RegistryGetString
//
// Purpose: used to read a REG_SZ registry value
```

```
//
// Arguments:
//   IN HKEY hKey - handle to a currently open key, or a predefined root value
//   IN LPCTSTR pszSubKey - the key under which the value is stored (if this
//                  value is null pszValueName is assumed to be under
//                  hKey)
//   IN LPCTSTR pszValueName - the name of the value to retrieve (may be NULL
//                    to retrieve the default value)
//   OUT LPTSTR pszData - a pointer to the buffer which will receive the
//                  data associated with the value
//   IN UINT cchData - a pointer to the variable which is length in characters
//                of the buffer pointed to by pData
//
// Return Values:
//   HRESULT
//   S_OK - the data was copied to the buffer
//   E_INVALIDARG - hKey or pszData is invalid
//   E_DATATYPE_MISMATCH - the value is not of type REG_SZ
//   E_INSUFFICIENT_BUFFER - the size of the buffer pointed to by pszData, as
//                  determined by cchData is not large enough to hold
//                  the string
//   An error value returned from RegOpenKeyEx or RegQueryValueEx wrapped as
//   a FACILITY_WIN32 HRESULT
//
// Results:
//   SUCCEEDED - pData points to the data associated with the value
//   FAILED - no change.
//
// Description:
//   the string associated with the value is copied to the buffer pointed to
```

```
//   by pData. If the key pointed to by hKey+pszSubKey does not exist,
//   RegistryGetString will fail since it uses RegOpenKey to access the key
//
// ************************************************************
HRESULT WINAPI RegistryGetString(HKEY hKey,
                                 LPCTSTR pszSubKey,
                                 LPCTSTR pszValueName,
                                 LPTSTR pszData,
                                 UINT cchData);


// ************************************************************
// Function Name: RegistrySetDWORD
//
// Purpose: used to set a REG_DWORD registry value
//
// Arguments:
//   IN HKEY hKey - handle to a currently open key, or a predefined root value
//   IN LPCTSTR pszSubKey - the key under which the value is stored (if this
//                          value is null pszValueName is assumed to be under
//                          hKey)
//   IN LPCTSTR pszValueName - the name of the value to set (may be NULL
//                             to set the default value)
//   IN DWORD dwData - the new value
//
// Return Values:
//   HRESULT
//   S_OK - the data value for pszValueName was changed to the data in dwData
//   E_INVALIDARG - hKey is invalid
//   E_DATATYPE_MISMATCH - the value is not of type REG_DWORD
```

```
//   An error value returned from RegOpenKeyEx or RegQueryValueEx wrapped as
//   a FACILITY_WIN32 HRESULT
//
// Results:
//   SUCCEEDED - the data associated with the value was changed
//   FAILED - no change
//
// Description:
//   the data associated with the value is changed to the new value. If the
//   key pointed to by hKey+pszSubKey does not exist, RegistrySetDWORD will
//   fail since it uses RegOpenKey to access the key
//
// ************************************************************
HRESULT WINAPI RegistrySetDWORD(HKEY hKey,
                    LPCTSTR pszSubKey,
                    LPCTSTR pszValueName,
                    DWORD dwData);


// ************************************************************
// Function Name: RegistrySetString
//
// Purpose: used to set a REG_SZ registry value
//
// Arguments:
//   IN HKEY hKey - handle to a currently open key, or a predefined root value
//   IN LPCTSTR pszSubKey - the key under which the value is stored (if this
//                  value is null pszValueName is assumed to be under
//                  hKey)
//   IN LPCTSTR pszValueName - the name of the value to set (may be NULL
```

```
//                  for to set the default value)
//   IN LPCTSTR pszData - the new value. This string is null terminated.
//
// Return Values:
//   HRESULT
//   S_OK - the data value for pszValueName was changed to the data in pszData
//   E_INVALIDARG - hKey or pszData is invalid
//   E_DATATYPE_MISMATCH - the value is not of type REG_SZ
//   An error value returned from RegOpenKeyEx or RegQueryValueEx wrapped as
//   a FACILITY_WIN32 HRESULT
//   An error HRESULT returned from StringCbLength
//
// Results:
//   SUCCEEDED - the data associated with the value was changed
//   FAILED - no change
//
// Description:
//   the data associated with the value is changed to the new value. If the
//   key pointed to by hKey+pszSubKey does not exist, RegistrySetString will
//   fail since it uses RegOpenKey to access the key
//
// ************************************************************
HRESULT WINAPI RegistrySetString(HKEY hKey,
                    LPCTSTR pszSubKey,
                    LPCTSTR pszValueName,
                    LPCTSTR pszData);


// ************************************************************
// Function Name: RegistryTestExchangeDWORD
```

```
//
// Purpose: used to atomically set a value based on a condition
//
// Arguments:
//   IN HKEY hKey - handle to a currently open key, or a predefined root value
//   IN LPCTSTR pszSubKey - the key under which the value is stored (if this
//                  value is null pszValueName is assumed to be under
//                  hKey)
//   IN LPCTSTR pszValueName - the name of the value to set (may be NULL
//                    to set the default value)
//   IN DWORD dwOldValue - the value to check against
//   IN DWORD dwNewValue - the value to set conditionally
//
// Return Values:
//   HRESULT
//   S_OK - the data value was changed to dwNewValue
//   S_FALSE - the target value was not set to dwNewValue because the DWORD
//         value assocated with pszValueName was not equal to dwOldValue
//   E_INVALIDARG - the handle or one of the pointers passed in was invalid
//   E_DATATYPE_MISMATCH - the value is not of type REG_DWORD
//   E_NOT_FOUND - the value could not be found under the specified key
//   An error value wrapped as a FACILITY_WIN32 HRESULT
//
// Results:
//   SUCCEEDED - the data value associated with pszData was dwOldValue and is
//           now dwNewValue
//   FAILED - no change
//
// Description:
//   This function is an interlocked function — in other words, it can be
```

```
//   considered atomic. It checks to see if the DWORD value associated with
//   pszValueName is equal to OldValue. If so, it sets the target to NewValue,
//   otherwise, it fails.
//
// ******************************************************
HRESULT WINAPI RegistryTestExchangeDWORD(HKEY hKey,
                                         LPCTSTR pszSubKey,
                                         LPCTSTR pszValueName,
                                         DWORD dwOldValue,
                                         DWORD dwNewValue);


// ******************************************************
// Function Name: RegistryNotifyApp
//
// Purpose: used to request that an app be launched or notified when a
//          specified value has been changed
//
// Arguments:
//   IN HKEY hKey - handle to a currently open key, or a predefined root value
//   IN LPCTSTR pszSubKey - the key under which the value is stored (if this
//                          value is null pszValueName is assumed to be under
//                          hKey)
//   IN LPCTSTR pszValueName - the name of the value on which change
//                             notifications are requested (may be NULL to
//                             indicate the default value)
//   IN LPCTSTR pszName - a user defined string representing this
//                        notification request, the string should be passed
//                        to RegistryStopNotification when notifications
//                        are no longer needed
```

```
//   IN LPCTSTR pszApp - pointer to string that is the path to the executable
//                to launch
//   IN LPCTSTR pszClass  - once the executable is launched, or if it is
//   IN LPCTSTR pszWindow   already running, a window with this window name and
//                class type is located in the process and the
//                notification is passed to it. If both of these
//                parameter are null this function will only launch
//                the application.
//   IN UINT msg - the message that will be passed to the window
//   IN NOTIFICATIONCONDITION * pCondition - the condition under which change
//                              notifications should be sent when
//                              a comparison of it to the new
//                              registry value is TRUE (may be
//                              NULL to indicate that any change
//                              should result in a notification)
//
// Return Values:
//   HRESULT
//   S_OK - the request for notification has been added to the notification
//        list
//   E_INVALIDARG - hKey, pszApp, or pszName is invalid
//   E_ALREADY_REGISTERED - a notification with a name equal to that specified
//                by pszName already exists
//   An error value wrapped as a FACILITY_WIN32 HRESULT
//
// Results:
//   SUCCEEDED - the caller will now be notified every time a change to this
//            value is made.
//   FAILED - no change
//
```

```
// Description:
//   The notification request is added to the notification list. The caller
//   calls RegistryStopNotification to stop further notifications. This
//   type of notification request is permanent, that is, the notification
//   request will be active even if the device is reset. On notification, this
//   function determines if an executable with the name specified in pszApp
//   is already running by doing a FindWindow on the class name and window name
//   specified by pszClass and pszWindow, if not it launches the app. The
//   command line passed to the application should be specified in the pszApp
//   string, in addition to any command line specified by the user, the
//   following is appended:
//      /notify "pszName"
//      pszName - the handle string passed to the function in the pszName
//                parameter
//   After the app is launched a FindWindow is done looking for a window with
//   the class name and window name specified in pszClass and pszWindow. If the
//   window is found, the message specified by msg is posted as in
//   RegistryNotifyWindow, via PostMessage. The parameters to the PostMessage
//   are as follows:
//      WPARAM - for values of type REG_DWORD this is the new value or 0
//               if the value was deleted; for all other types this value
//               is 0
//      LPARAM - 0
//   The msg parameter should be unique for each call to RegistryNotifyApp
//   so that the client can differentiate between the multiple notifications.
//   The client will be notified when the value is added as well as on changes.
//   When a notification arrives, if the application pointed to by pszApp can
//   not be launched or a window with a class of type pszClass can not be found
//   or the PostMessage fails, the notification will be removed from the
//   notification list.
```

```
//
// ****************************************************************
HRESULT WINAPI RegistryNotifyApp(HKEY hKey,
                    LPCTSTR pszSubKey,
                    LPCTSTR pszValueName,
                    LPCTSTR pszName,
                    LPCTSTR pszApp,
                    LPCTSTR pszClass,
                    LPCTSTR pszWindow,
                    UINT msg,
                    NOTIFICATIONCONDITION * pCondition);


// ****************************************************************
// Function Name: RegistryNotifyWindow
//
// Purpose: used to request that a window be notified when a specified value
//       has been changed
//
// Arguments:
//   IN HKEY hKey - handle to a currently open key, or a predefined root value
//   IN LPCTSTR pszSubKey - the key under which the value is stored (if this
//                  value is null pszValueName is assumed to be under
//                  hKey)
//   IN LPCTSTR pszValueName - the name of the value on which change
//                    notifications are requested (may be NULL to
//                    indicate the default value)
//   IN HWND hWnd - the handle of the window to which the message will be sent
//   IN UINT msg - the message that will be passed to the window
//   IN DWORD dwUserData - user data that will be passed back to the user
```

```
//                  with the notification
//   IN NOTIFICATIONCONDITION * pCondition - the condition under which change
//                              notifications should be sent when
//                              a comparison of it to the new
//                              registry value is TRUE (may be
//                              NULL to indicate that any change
//                              should result in a notification)
//   OUT HREGNOTIFY * phNotify - receives the handle to the notification
//                       request. This handle should be closed using
//                       RegistryCloseNotification when notifications
//                       on this key are no longer needed.
//
// Return Values:
//   HRESULT
//   S_OK - the request for notification has been added to the notification
//        list
//   E_INVALIDARG - hKey, phNotify or hWnd is invalid
//   An error value wrapped as a FACILITY_WIN32 HRESULT
//
// Results:
//   SUCCEEDED - the caller will now be notified every time a change to this
//            value is made.
//   FAILED - no change
//
// Description:
//   The msg parameter should be unique for each call to RegistryNotifyWindow
//   so that the client can differentiate between the multiple notifications.
//   The notification request is added to the notification list. The caller
//   calls RegistryCloseNotification to stop further notifications and to
//   close the notification handle. This type of notification request is
```

```
//   transient, that is, if the device is reset the notification request will
//   no longer exist. When the value specified by pszValueName is changed, the
//   client is notified via a PostMessage. If the PostMessage fails or the
//   window specified by hWnd is no longer valid the notification request is
//   removed from the notification queue and the handle returned in phNotify
//   is closed.
//   The parameters passed on the PostMessage are as follows:
//     WPARAM - for values of type REG_DWORD this is the new value or 0
//           if the value was deleted; for all other types this value is 0
//     LPARAM - the value passed in on dwUserData
//   If the value does not exist at the time of the call to
//   RegistryNotifyWindow, the client will be notified when the value is added.
//
// ***************************************************************
HRESULT WINAPI RegistryNotifyWindow(HKEY hKey,
                        LPCTSTR pszSubKey,
                        LPCTSTR pszValueName,
                        HWND hWnd,
                        UINT msg,
                        DWORD dwUserData,
                        NOTIFICATIONCONDITION * pCondition,
                        HREGNOTIFY * phNotify);


// ***************************************************************
// Function Name: RegistryNotifyMsgQueue
//
// Purpose: used to request that a message queue be notified when a specified
//       value has been changed
//
```

```
// Arguments:
//   IN HKEY hKey - handle to a currently open key, or a predefined root value
//   IN LPCTSTR pszSubKey - the key under which the value is stored (if this
//                  value is null pszValueName is assumed to be under
//                  hKey)
//   IN LPCTSTR pszValueName - the name of the value on which change
//                    notifications are requested (may be NULL to
//                    indicate the default value)
//   IN LPCTSTR pszMsgQueue - A pointer to a string that is the name of the
//                    message queue to notify. If this message queue
//                    has not yet been created, RegistryNotifyMsgQueue
//                    will create it.
//   IN DWORD dwUserData - user data that will be passed back to the user
//                  with the notification
//   IN NOTIFICATIONCONDITION * pCondition - the condition under which change
//                              notifications should be sent when
//                              a comparison of it to the new
//                              registry value is TRUE (may be
//                              NULL to indicate that any change
//                              should result in a notification)
//   OUT HREGNOTIFY * phNotify - receives the handle to the notification
//                      request. This handle should be closed using
//                      RegistryCloseNotification when notifications
//                      on this key are no longer needed.
//
// Return Values:
//   HRESULT
//   S_OK - the request for notification has been added to the notification
//        list
//   E_INVALIDARG - hKey, phNotify, or pszMsgQueue is invalid
```

```
//    An error value wrapped as a FACILITY_WIN32 HRESULT
//
// Results:
//    SUCCEEDED - the caller will now be notified every time a change to this
//                value is made.
//    FAILED - no change
//
// Description:
//    The notification request is added to the notification list. The caller
//    calls RegistryCloseNotification to stop further notifications and
//    close the notification handle. This type of notification request is
//    transient, that is, if the device is reset the notification request will
//    no longer exist. The client will be notified of changes via the message
//    queue. Once the client is notified, information about the changed key can
//    be retrieved using the ReadMsgQueue function. The data returned is in the
//    following format:
//      | HREGNOTIFY hNotify | DWORD dwUserData | UINT byte count | BYTE[]
new value |
//      hNotify - the same hNotify as returned in phNotify
//      dwUserData - the user data passed as dwUserData
//      byte count - the number of bytes to follow (for deletes this value is 0)
//      new value - the new value for pszValueName
//    If the value does not exist at the time of the call to
//    RegistryNotifyMsgQueue, the client will be notified when the value is
//    added.
//
// ************************************************************
HRESULT WINAPI RegistryNotifyMsgQueue(HKEY hKey,
                                      LPCTSTR pszSubKey,
                                      LPCTSTR pszValueName,
```

```
    LPCTSTR pszMsgQueue,
    DWORD dwUserData,
    NOTIFICATIONCONDITION * pCondition,
    HREGNOTIFY * phNotify);


// ***************************************************************
// Function Name: RegistryNotifyCallback
//
// Purpose: used to request that a callback be notified when a specified
//       value has been changed
//
// Arguments:
//   IN HKEY hKey - handle to a currently open key, or a predefined root value
//   IN LPCTSTR pszSubKey - the key under which the value is stored (if this
//                  value is null pszValueName is assumed to be under
//                  hKey)
//   IN LPCTSTR pszValueName - the name of the value on which change
//                    notifications are requested (may be NULL to
//                    indicate the default value)
//   IN REGISTRYNOTIFYCALLBACK pfnRegistryNotifyCallback - A pointer to a
//                                      function that will
//                                      be called back when
//                                      a notification
//                                      arrives
//   IN DWORD dwUserData - user data that will be passed back to the user
//                  with the notification
//   IN NOTIFICATIONCONDITION * pCondition - the condition under which change
//                              notifications should be sent when
//                              a comparison of it to the new
```

```
//                          registry value is TRUE (may be
//                          NULL to indicate that any change
//                          should result in a notification)
//   OUT HREGNOTIFY * phNotify - receives the handle to the notification
//                       request. This handle should be closed using
//                       RegistryCloseNotification when notifications
//                       on this key are no longer needed.
//
// Return Values:
//   HRESULT
//   S_OK - the request for notification has been added to the notification
//        list
//   E_INVALIDARG - hKey, phNotify, or pfnRegistryNotifyCallback is invalid
//   An error value wrapped as a FACILITY_WIN32 HRESULT
//
// Results:
//   SUCCEEDED - the caller will now be notified every time a change to this
//             value is made.
//   FAILED - no change
//
// Description:
//   The notification request is added to the notification list. The caller
//   calls RegistryCloseNotification to stop further notifications and
//   close the notification handle. This type of notification request is
//   transient, that is, if the device is reset the notification request will
//   no longer exist. The client will be notified of changes via the callback.
//   If the value does not exist at the time of the call to
//   RegistryNotifyCallback, the client will be notified when the value is
//   added.
//
```

```
//
**********************************************************************
****
HRESULT WINAPI RegistryNotifyCallback(HKEY hKey,
                        LPCTSTR pszSubKey,
                        LPCTSTR pszValueName,
                        REGISTRYNOTIFYCALLBACK pfnRegistryNotifyCallback,
                        DWORD dwUserData,
                        NOTIFICATIONCONDITION * pCondition,
                        HREGNOTIFY * phNotify);


// ************************************************************
// Function Name: RegistryCloseNotification
//
// Purpose: used remove a request for notifications from the notification list
//        and close the notification handle
//
// Arguments:
//   IN HREGNOTIFY hNotify - the handle to a valid HREGNOTIFY, must have been
//                    returned one of the RegistryNotify* functions
//
// Return Values:
//   HRESULT
//   S_OK - the notification request was removed from the list
//   E_INVALID_HANDLE - hNotify is invalid
//   An error value wrapped as a FACILITY_WIN32 HRESULT
//
// Results:
//   SUCCEEDED - the notification request has been removed, hNotify is now
```

```
//              an invalid handle
//   FAILED - no change
//
// Description:
//   the notification list is searched for references to hNotify, if found
//   they are removed the queue and the memory associated with hNotify is
//   freed. Any notifications that have not yet been dispatched to the client
//   will be lost.
//
// ****************************************************************
HRESULT WINAPI RegistryCloseNotification(HREGNOTIFY hNotify);


// ****************************************************************
// Function Name: RegistryStopNotification
//
// Purpose: used to remove a request for a permanent notification from
//          the notification list
//
// Arguments:
//   IN LPCTSTR pszName - a string representing a permanent notification
//                        that has already been registered
//
// Return Values:
//   HRESULT
//   S_OK - the notification request was removed from the list
//   E_INVALIDARG - pszName is invalid
//   An error value wrapped as a FACILITY_WIN32 HRESULT
//
// Results:
```

```
//   SUCCEEDED - the notification request has been removed
//   FAILED - no change
//
// Description:
//   The notification list is searched for references to the notification,
//   if found they are removed the queue. Any notifications that have not
//   yet been dispatched to the client are lost.
//
// ****************************************************************
HRESULT WINAPI RegistryStopNotification(LPCTSTR pszName);


// ****************************************************************
// Function Name: RegistryBatchNotification
//
// Purpose: used to batch frequently occurring notifications
//
// Arguments:
//   IN HREGNOTIFY hNotify - the handle to a valid HREGNOTIFY, has been
//                  returned one of the RegistryNotify* functions
//   IN DWORD dwMillisecondsIdle - the number of milliseconds the value should
//                      be constant before the notification is fired
//   IN DWORD dwMillisecondsMax - the maximum number of milliseconds between
//                      the time the first change happens and the
//                      notification is sent
//
// Return Values:
//   HRESULT
//   S_OK - the notification request was removed from the list
//   E_INVALIDARG - dwMillisecondsIdle is set to INFINITE
```

```
//   E_INVALID_HANDLE - hNotify is invalid
//   An error value wrapped as a FACILITY_WIN32 HRESULT
//
// Results:
//   SUCCEEDED - the new batch times are set
//   FAILED - no change
//
// Description:
//   when a value is changed, the notification system waits
//   dwMillisecondsIdle milliseconds and then sends the notification. If a
//   another change happens during that wait period the timer is reset and
//   the notification system will wait another dwMillisecondsIdle
//   milliseconds to send the notification. In order to ensure that the
//   dwMillisecondsIdle doesn't prevent the notification from ever being
//   sent, dwMillisecondsMax is used. dwMillisecondsMax is the maximum
//   number of milliseconds that can pass from the time the first notification
//   arrives and the time notification is sent. If dwMillisecondsMax is set to
//   INFINITE the notification will batch until the value is idle.
//   dwMillisecondsIdle can not be INFINITE.
//
// ************************************************************
HRESULT WINAPI RegistryBatchNotification(HREGNOTIFY hNotify,
                                         DWORD dwMillisecondsIdle,
                                         DWORD dwMillisecondsMax);

HRESULT WINAPI RegistryScheduleNotifications(SYSTEMTIME * pstStart,
                                             DWORD dwIntervalMinutes,
                                             RSN_BEHAVIOR rsnBehavior,
                                             LPCTSTR pszName,
                                             LPCTSTR pszApp,
```

```
                    LPCTSTR pszClass,
                    LPCTSTR pszWindow,
                    UINT msg);
```

Managed API

The following is an exemplary Managed API:

```
namespace A.Mobile
{
/// used to specify what should happen when an event occurs while device is in standby
        public enum StandbyBehavior
    {
        /// do not raise the event at all
        Ignore,
        /// bring the device out of standby and raise the event
        Wake,
        /// raise the event once the device is woken up by something else
        Delay
    }

    public interface IAppLaunchable
    {
        /// Unique name for this notification
        public string appLaunchId{ get; }

    /// register this notification for a specific executable and command line params
        /// until this is called, the notification is not active

        /// <param name="appLaunchId">
        /// unique identifier for this notification.
```

```
        /// used to open it back up when the application closes/restarts
        /// an exception is thrown if this ID is already in use
        /// </param>
        /// <param name="filename">
        /// application to launch when notification is raised.
        /// if null or empty, the calling executable is used
        /// </param>
        /// <param name="parameters">command line parameter to send to
application</param>
        void EnableAppLaunch( string appLaunchId, string filename, string
parameters );
        /// register this notification for a specific executable with no command
line params
        /// until this is called, the notification is not active
        /// <param name="appLaunchId">
        /// unique identifier for this notification.
        /// used to open it back up when the application closes/restarts
        /// an exception is thrown if this ID is already in use
        /// </param>
        /// <param name="filename">application to launch when notification is
raised</param>
        void EnableAppLaunch( string appLaunchId, string filename);
        /// register this notification for the calling executable with no parameters
        /// until this is called, the notification is not active.
        ///
        /// if this is called from a DLL rather than an EXE, an exception is
thrown.
        /// DLLs need to call one of the other overloads
        /// <param name="appLaunchId">
        /// unique identifier for this notification.
```

```
            /// used to open it back up when the application closes/restarts
            /// an exception is thrown if this ID is already in use
            /// </param>
            void EnableAppLaunch( string appLaunchId );
            /// unregistr this notification
            /// used to stop the notification from firing
            void DisableAppLaunch();
        }
        /// notification that raises on a scheduled basis
            public class ScheduledNotification : ILaunchable
        {
            // when the next occurence of this notification is scheduled to occur
            public DateTime NextOccurrence{ get; set; }

            /// how much time passes between occurrences of this notification
            /// seconds and milliseconds are ignored
            public TimeSpan Interval{ get; set; }

            /// how this notification acts when the device is in standby during an
occurrence
            /// ignore it, wake the device, wait until something else wakes the device.
            /// default value is Wake.
            public StandbyBehavior StandbyBehavior{ get; set; }

            /// determines if the named notification is registered
            /// <param name="appLaunchId">name of a notification</param>
            /// <returns>true if the notification is registered</returns>
            public static bool AppLaunchIdExists( string appLaunchId ) { }

            public event EventHandler Occurred;
```

```
            /// create a new persistent notification with the given name
            /// <param name="nextOccurrence">when the next occurrence
is</param>
            /// <param name="interval">how long between occurrences.  seconds
and milliseconds are ignored</param>
            public ScheduledNotification( DateTime nextOccurrence, TimeSpan
interval ) { }
            /// constructor that loads in a previously registered notification
            /// <param name="existingAppLaunchId">name of previously registered
notification</param>
            public ScheduledNotification( string existingAppLaunchId )
            {
                  if (!AppLaunchIdExists( name ) )
                        throw new ArgumentException();
            }
      }

}

namespace A. Mobile.Status
{
      /// used for conditional change notifications to specify how the new value
      /// should be compared to the desired value
      public enum StateComparisonType
      {
            /// event is raised regardless of value.  this is the default
            All,
            Equal,
            NotEqual,
            Greater,
```

```
            GreaterOrEqual,
            LessOrEqual,
            Less,
            Contains,
            StartsWith,
            EndsWith
        }
        /// Enum that represents all of the system states that can be queried and listened
to.

        public abstract class StateBase : ILaunchable
        {
            /// for conditional notifications, how to compare new value to
TargetValue
            public ComparisonType ComparisonType { get; set; }

            /// what to compare new value to.  notification only raises if comparison
is true
            public object ComparisonValue { get; set; }
            /// current value of this system property
            public object CurrentValue { get; }

            public event ChangeNotificationEventHandler Changed;
        }
        /// transient notification that raises when a system-defined property changes
        public class SystemState : StateBase
        {
            /// gets the value of the specified system property
            /// <param name="property">property to get the value of</param>
            /// <returns></returns>
```

```
        public static object GetValue( SystemProperty property ) { }

        /// system property to monitor
        public SystemProperty Property { get; }
        /// determines if the named notification is registered
        /// <param name="appLaunchId">name of a notification</param>
        /// <returns>true if the notification is registered</returns>
        public static bool AppLaunchIdExists( string appLaunchId ) { }

        /// constructor with no conditionals (event is always raised)
        /// <param name="property"></param>
        public SystemState( SystemProperty property ) { }
        /// constructor thats sets conditionals for when event should be raised
        /// <param name="property">property to watch</param>
        /// <param name="comparisonType">how to compare it</param>
        /// <param name="comparisonValue">what to compare it to</param>
        public SystemState( SystemProperty property, ComparisonType
comparisonType, object comparisonValue ) { }

        /// constructor that loads in a previously registered notification
        /// <param name="existingAppLaunchId">name of previously registered
notification</param>
        public SystemState( string existingAppLaunchId )
        {
                if (!AppLaunchIdExists( name ) )
                        throw new ArgumentException();
        }


    }
```

```
/// transient notification that raises when a registry value changes
public class RegistryValue : StateBase
{
        /// registry key to monitor
        public RegistryKey Key { get; }

        /// name of value in registry key to monitor
        public string ValueName { get; }

        /// determines if the named notification is registered
        /// <param name="appLaunchId">name of a notification</param>
        /// <returns>true if the notification is registered</returns>
        public static bool AppLaunchIdExists( string appLaunchId ) { }

        /// constructor with no conditionals (event is always raised)
        /// <param name="key">registry key to watch</param>
        /// <param name="valueName">name of registry value in the key to
watch</param>
        public RegistryValue( RegistryKey key, string valueName ) { }

        /// constructor thats sets conditionals for when event should be raised
        /// <param name="key">registry key to watch</param>
        /// <param name="valueName">name of registry value in the key to
watch</param>
        /// <param name="comparisonType">how to compare it</param>
        /// <param name="comparisonValue">what to compare it to</param>
        public RegistryValue( RegistryKey key, string valueName,
ComparisonType comparisonType, object comparisonValue ) { }

        /// constructor that loads in a previously registered notification
```

```
        /// <param name="existingAppLaunchId">name of previously registered
notification</param>
        public RegistryValue( string existingAppLaunchId )
        {
            if (!AppLaunchIdExists( name ) )
                throw new ArgumentException();
        }
    }

    public delegate void ChangeNotificationEventHandler( object sender,
ChangeNotificationEventArgs e );
    public class ChangeNotificationEventArgs : EventArgs
    {
        public object CurrentValue { get; }
    }
}
```

Sample Usage of Managed API

```
// querying a system property
int signal = SystemState.PhoneSignalStrength;
// or...
int signal = (int)SystemState.GetValue( SystemProperty.PhoneSignalStrength );
// or...
SystemState state = new SystemState( SystemProperty.PhoneSignalStrength );
int signal = (int)state.CurrentValue;
```

```
// registering for a transient notification
SystemState state = new SystemState( SystemProperty.PhoneSignalStrength );
```

```
state.Changed += new ChangeNotificationEventHandler( ... );
```

```
// registering for a persistent notification with conditional
SystemState state;
if( SystemState.AppLaunchIdExists( "MyApp.GoodSignal" ) )
{
        state = new SystemState( "MyApp.GoodSignal" );
}
else
{
        state = new SystemState( SystemProperty.PhoneSignalStrength,
ComparisonType.Greater, 75 );
        state.EnableAppLaunch( "MyApp.GoodSignal" );
}
state.Changed += new ChangeNotificationEventHandler( ... );
```

```
// registering for a scheduled notification
ScheduledNotification daily;
if( ScheduledNotification.AppLaunchIdExists( "MyApp.Daily" ) )
{
        daily= new ScheduledNotification( "MyApp.Daily" );
}
else
{
        daily = new ScheduledNotification( DateTime.Now, new TimeSpan( 24, 0, 0 ) );
        daily.EnableAppLaunch( "MyApp.Daily" );
}
                daily.Occurred += new EventHandler( ... );
```

The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

What is claimed is:

1. A method for providing notifications to clients in response to state property changes, comprising:

at a notification broker that is located on a mobile device, receiving an automatic notification request from a client application on the mobile device to receive a notification in response to an event that originates on the mobile device; wherein the event is associated with change in a state property of the mobile device, wherein execution of the client application on the mobile device is dependent upon a received notification; wherein the notification request comprises a permanent notification request and a transient notification request; wherein the permanent notification request is stored by the notification broker in a data-store on the mobile device that maintains the notification request across a reboot; wherein the transient notification is not maintained across the reboot;

ensuring that the state property is registered with the notification broker, wherein the notification broker includes state properties that are updated by different components within the mobile device;

determining when the state property changes;

determining when the client should receive notification of the state property change; and

the notification broker on the mobile device notifying the client of the state property change when determined.

2. The method of claim **1**, further comprising utilizing an Application Program Interface (API) to perform actions involving the state properties, wherein the actions include at least one of the following: registering a state property; querying the state property; and setting the state property.

3. The method of claim **2**, wherein determining when the client should receive the notification, comprises: applying a conditional expression to the state property and notifying the client of the state property change when the condition is met.

4. The method of claim **3**, wherein the conditional expression is expressed using the following conditions: all, equal, not equal, greater than, greater or equal than, less than or equal, less than, contains, starts with, and ends with.

5. The method of claim **2**, further comprising launching the client in response to at least one of: a state property change and a scheduled event.

6. The method of claim **2**, further comprising notifying the client in response to a schedule defined by the client.

7. The method of claim **2**, wherein determining when the state property changes further comprises: performing a batching operation on changes to the state property that occur within a predetermined time period.

8. The method of claim **1**, wherein receiving the notification request from the client to receive the notification in response to the change associated with the state property, further comprises associating a group of state properties with the notification request from the client when an identifier associated with the request identifies a category of state properties, wherein the state properties are arrange in a hierarchy within the notification system.

9. The method of claim **2**, further comprising providing pre-loaded state property data that may be accessed by the clients without the client having to register a state property with the notification broker; wherein the pre-loaded state property data comprises a majority of the following states; a display orientation state; an undismissed reminders state; an undismissed alarms state; a battery state; a memory state; a storage card state; a hardware state, a keyboard enabled state, a Wifi enabled state, a Bluetooth enabled state, a headphones state, a camera state; a messaging state; a tasks state; a calendar state; an Instant Messenger state; a connectivity state; a media player state; a synchronization status state; and a telephony state.

10. A system for state management and notifications, comprising:

a data store on a mobile device that is arranged to store information relating to state properties, wherein at least some of the state properties are modified by different components;

an Application Program Interface (API) configured to perform operations relating to the state properties;

client applications on the mobile device that are configured to automatically register notification requests and receive notifications in response to a change in a state property of the mobile device for which they have registered, wherein the notification requests indicate when the clients should receive notifications in response to changes associated with the state properties, and wherein execution of the client applications is dependent upon a received notification; wherein the change in the state property is responsive to an event that originates on the mobile device;

a notification list stored within the data store that is arranged to store the clients that have been registered to receive notification requests;

a notification broker on the mobile device that is coupled to the data store, the notification list, and the clients, wherein the notification broker, includes functionality configured to perform the following actions, including to:

receive a notification request to add at least one client to the notification list;

add the at least one client to the notification list; and

determine when a registered state property changes, and when the state property changes, determine the clients to receive a notification, and notify the determined clients of the state property change.

11. The system of claim **10**, wherein the Application Program Interface (API) is further configured to perform at least one of the following actions: registering a state property; querying the state property; and setting the state property.

12. The system of claim **11**, wherein determining the clients to receive the notification, comprises: applying a conditional expression to the state property and notifying the client of the state property when the condition is met.

13. The system of claim **12**, wherein the conditional expression includes at least one of the following conditions: all, equal, not equal, greater than, greater or equal than, less than or equal, less than, contains, starts with, and ends with.

14. The system of claim **11**, further comprising launching the client in response to at least one of a state property change and a scheduled event.

15. The system of claim **11**, further comprising notifying the client in response to a schedule defined by the client.

16. The system of claim **11**, wherein determining when the state property changes further comprises: performing a batching operation on changes to the state property that occur within a predetermined time period.

17. The system of claim **10**, wherein the state properties are arranged in a hierarchy.

18. The system of claim **10**, wherein content within the data store persists across device reboots.

19. A tangible computer-readable storage medium having computer executable instructions for performing operations on state properties, comprising:

receiving at a notification broker on a mobile device an automatic request from a client application that is executed on the mobile device to receive an identifier identifying at least one state property within a group of state properties of the mobile device such that the execution of the application on the mobile device is in response to the state properties associated with the mobile device on which the application resides; wherein the application on the mobile device is developed after development of the mobile device; wherein state properties within the group of state properties are updated by different components on the mobile device; wherein the state properties change is response to an event that originates on the mobile device;

receiving the identifier at the mobile device that indicates the at least one state property within a group of state properties of the mobile device;

determining an operation to perform on the mobile device relating to a state property within the group of state properties; and

performing the operation on the mobile device in response to the received identifier.

20. The computer-readable medium of claim 19, wherein the operation includes at least one of the following operations: registering a state property; querying the state property; associating a conditional expression with the state property; and setting the state property.

21. The computer-readable medium of claim 20, wherein performing the operation, comprises: applying the conditional expression to the state property and notifying the client of the state property when the condition is met.

22. The computer-readable medium of claim 21, wherein the conditional expression includes at least one of the following conditions: all, equal, not equal, greater than, greater or equal than, less than or equal, less than, contains, starts with, and ends with.

23. The computer-readable medium of claim 20, wherein performing the operation further comprises launching a client application in response to at least one of the following: a change in the state property and a scheduled event.

24. The computer-readable medium of claim 20, wherein performing the operation further comprises notifying the client in response to a schedule defined by the client.

25. The computer-readable medium of claim 20, wherein performing the operation further comprises: performing a batching operation on changes to the state property that occur within a predetermined time period.

* * * * *

US005664133A

United States Patent [19]

Malamud et al.

[11] **Patent Number: 5,664,133**

[45] **Date of Patent: Sep. 2, 1997**

[54] **CONTEXT SENSITIVE MENU SYSTEM/ MENU BEHAVIOR**

[75] Inventors: **Mark A. Malamud**, Seattle; **John E. Elsbree**, Everett; **Laura J. Butler**, Bellevue; **David A. Barnes, Jr.**, Seattle, all of Wash.

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

[21] Appl. No.: **648,807**

[22] Filed: **Apr. 30, 1996**

Related U.S. Application Data

[63] Continuation of Ser. No. 166,339, Dec. 13, 1993, abandoned.

[51] **Int. Cl.**[6] **G06F 15/00**

[52] **U.S. Cl.** **345/352**; 345/339; 345/396; 345/146

[58] **Field of Search** 395/326, 333, 395/335, 339, 340–343, 346–348, 352–354; 345/146, 145

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,789,962	12/1988	Berry et al.	395/338
5,157,768	10/1992	Hoeber et al.	395/338
5,204,947	4/1993	Bernstein et al.	395/157
5,241,645	8/1993	Cimral et al.	395/500
5,243,697	9/1993	Hoeber et al.	395/334
5,249,300	9/1993	Bachman et al.	395/800
5,297,249	3/1994	Bernstein et al.	395/356
5,347,628	9/1994	Brewer et al.	395/351
5,581,686	12/1996	Koppolu et al.	395/340

OTHER PUBLICATIONS

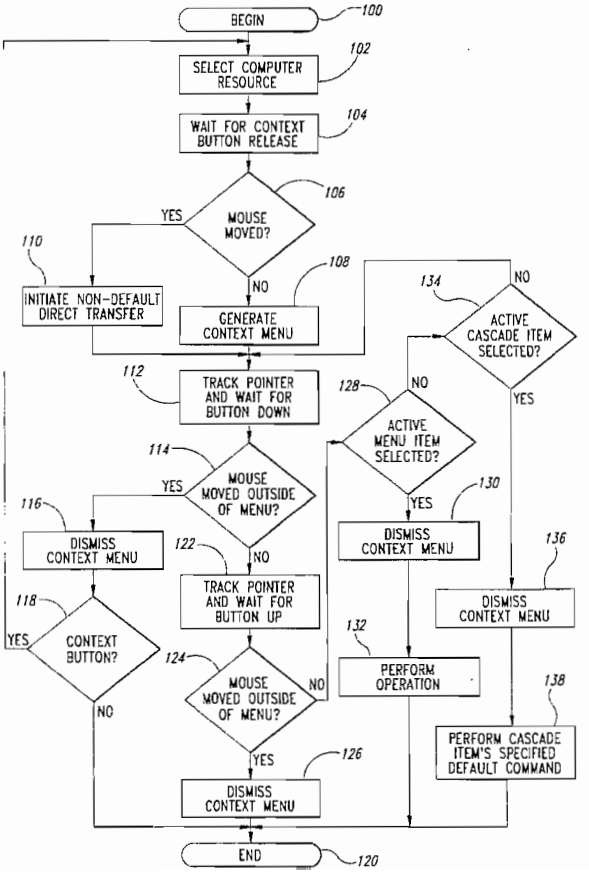
Microsoft Windows v. 3.1, Microsoft Corporation, 1985–1992.

Primary Examiner—Kee M. Tung
Assistant Examiner—U. Chauhan
Attorney, Agent, or Firm—Seed and Berry LLP

[57] **ABSTRACT**

A method and system are described for a computer system for retrieving and presenting a set of commands in the form of a pop up context menu for a selected object. The context menu is displayed in the proximity of the selected object and is determined primarily by the class of the selected object and secondarily by the particular container in which the selected object resides at the time of selection. The context menu displays a number of useful features which enable the user to quickly and easily invoke commands upon the selected object.

38 Claims, 9 Drawing Sheets



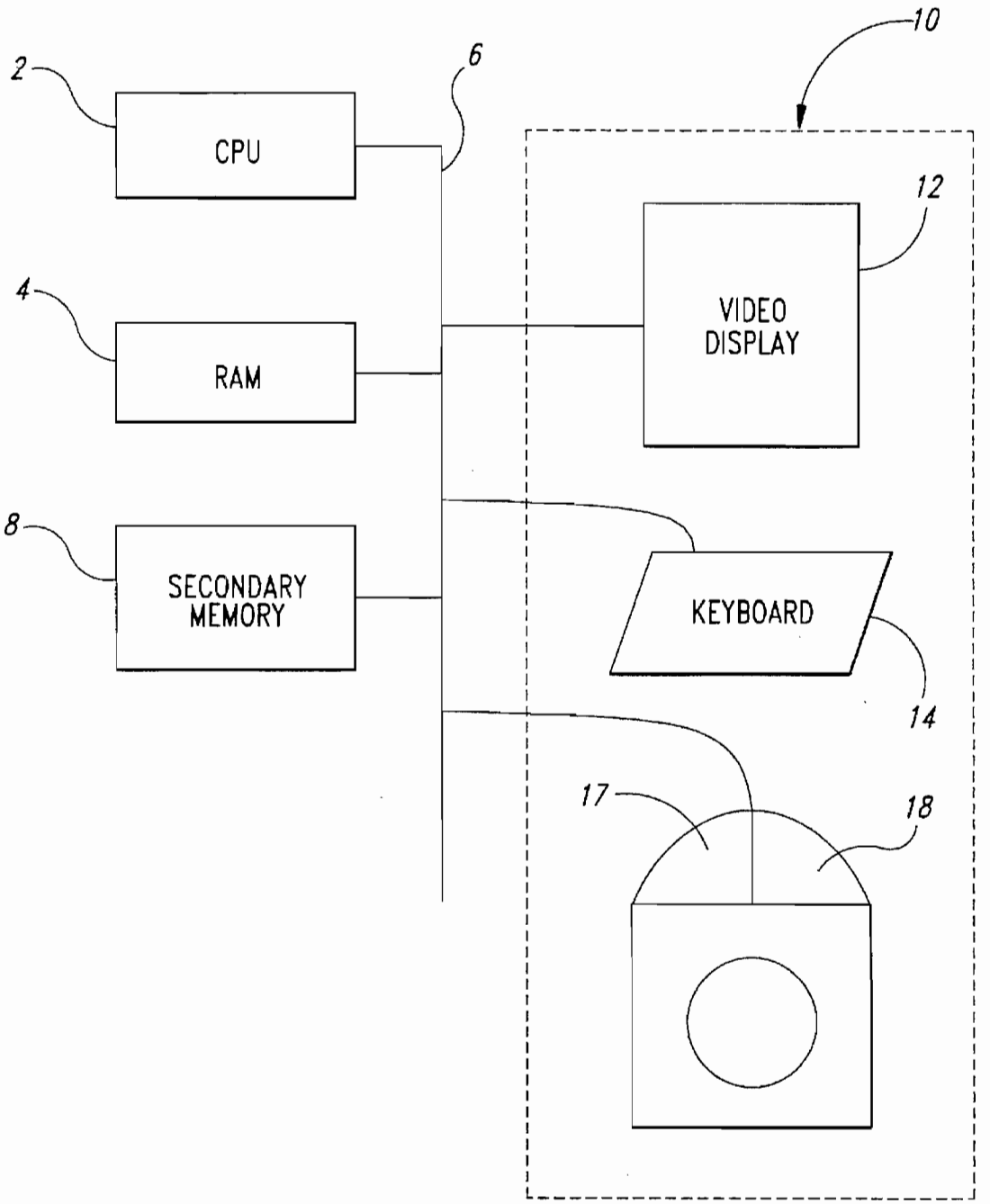

10
2
6
CPU
12
VIDEO DISPLAY
4
RAM
8
SECONDARY MEMORY
KEYBOARD
14
17
18

Fig. 1

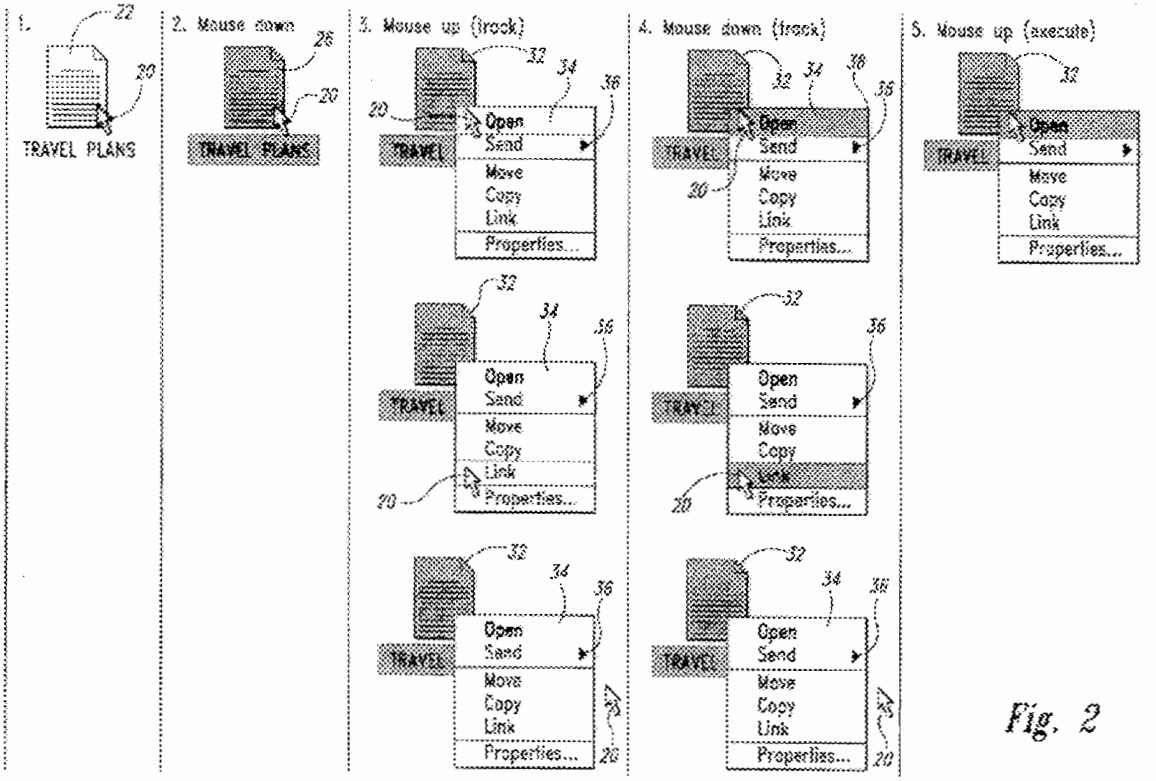

1.
2. Mouse down
3. Mouse up (track)
4. Mouse down (track)
5. Mouse up (execute)
TRAVEL PLANS
TRAVEL
Open
Send
Move
Copy
Link
Properties...

Fig. 2

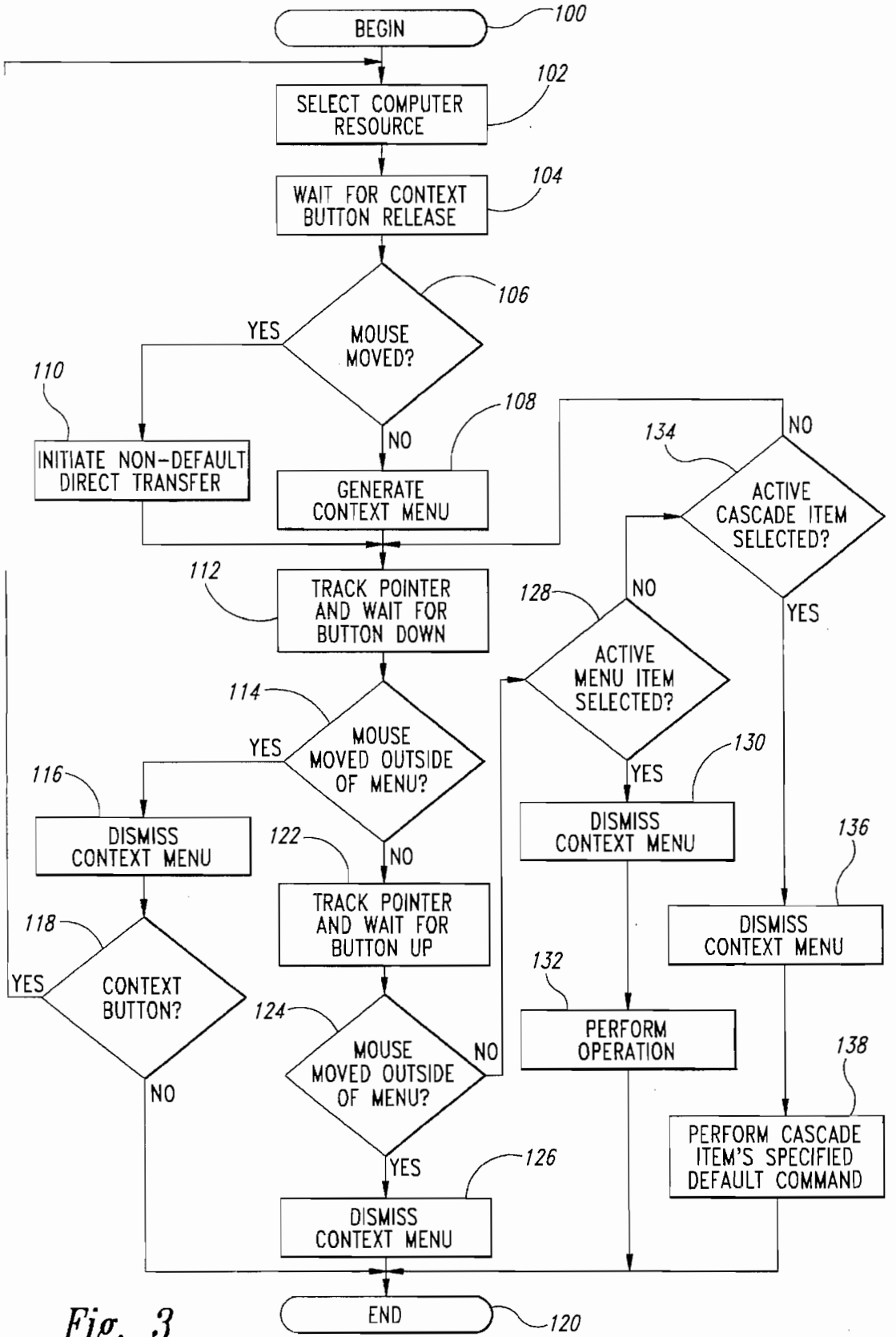

BEGIN
100
SELECT COMPUTER RESOURCE
102
WAIT FOR CONTEXT BUTTON RELEASE
104
MOUSE MOVED?
106
YES
NO
INITIATE NON-DEFAULT DIRECT TRANSFER
110
GENERATE CONTEXT MENU
108
TRACK POINTER AND WAIT FOR BUTTON DOWN
112
MOUSE MOVED OUTSIDE OF MENU?
114
YES
NO
DISMISS CONTEXT MENU
116
TRACK POINTER AND WAIT FOR BUTTON UP
122
CONTEXT BUTTON?
118
YES
NO
MOUSE MOVED OUTSIDE OF MENU?
124
NO
YES
DISMISS CONTEXT MENU
126
ACTIVE CASCADE ITEM SELECTED?
134
NO
YES
ACTIVE MENU ITEM SELECTED?
128
NO
YES
DISMISS CONTEXT MENU
130
PERFORM OPERATION
132
DISMISS CONTEXT MENU
136
PERFORM CASCADE ITEM'S SPECIFIED DEFAULT COMMAND
138
END
120

Fig. 3

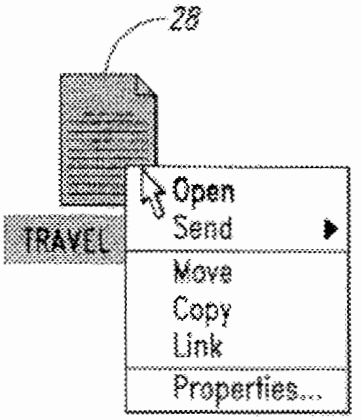

28
TRAVEL
Open
Send
Move
Copy
Link
Properties...

Fig. 4A

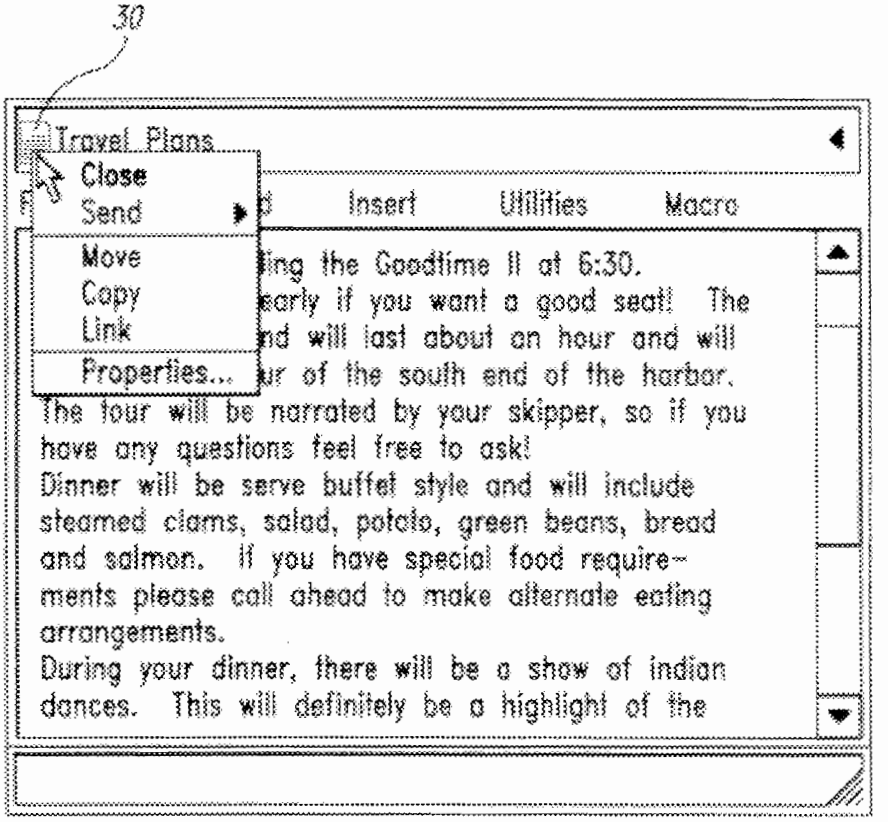

30
Travel Plans
Close
Send
Move
Copy
Link
Properties...
Insert
Utilities
Macro
ing the Goodtime II at 6:30.
early if you want a good seat! The
nd will last about an hour and will
ur of the south end of the harbor.
The tour will be narrated by your skipper, so if you
have any questions feel free to ask!
Dinner will be serve buffet style and will include
steamed clams, salad, potato, green beans, bread
and salmon. If you have special food require-
ments please call ahead to make alternate eating
arrangements.
During your dinner, there will be a show of Indian
dances. This will definitely be a highlight of the

Fig. 4B

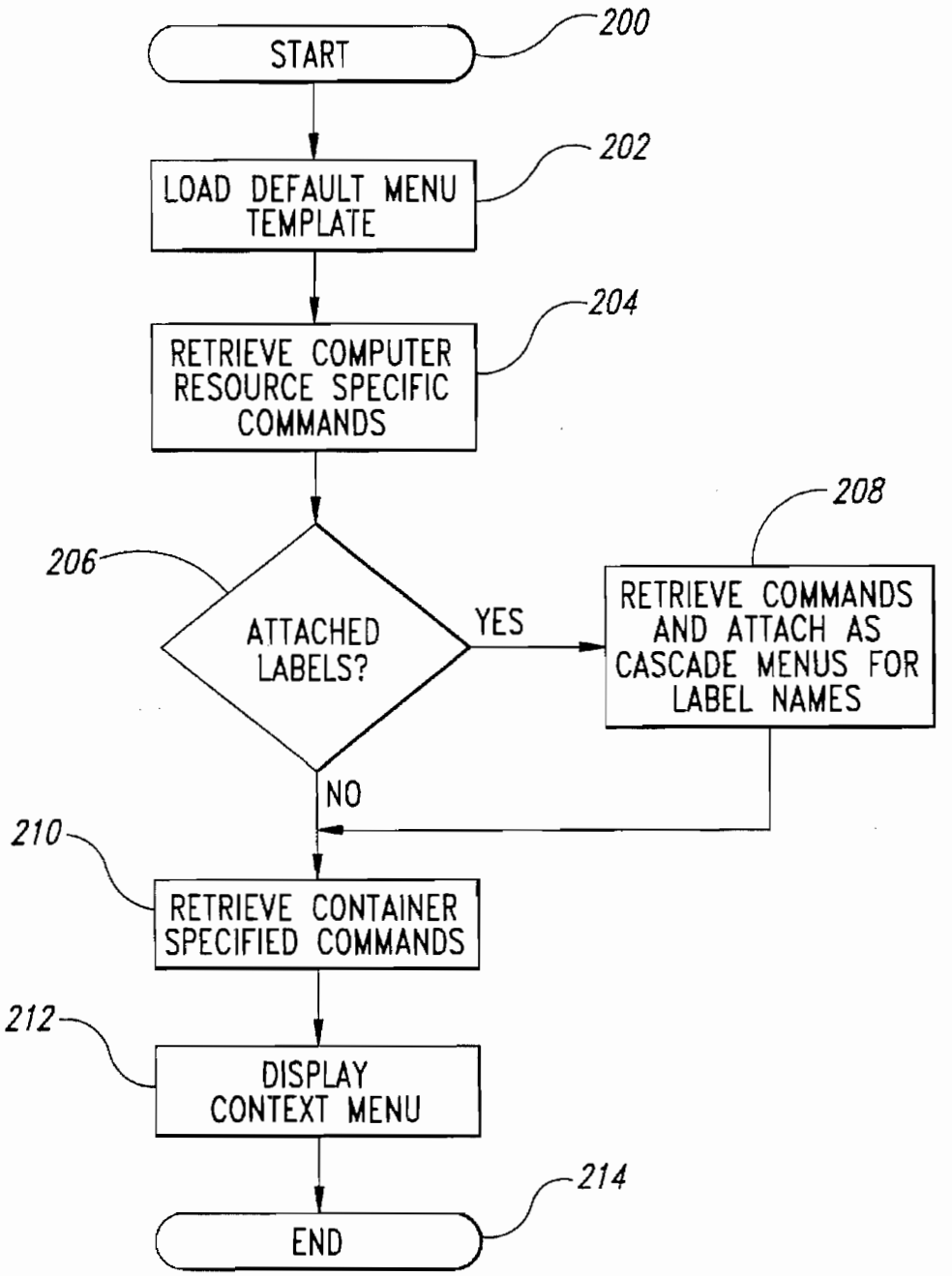

START
200
LOAD DEFAULT MENU TEMPLATE
202
RETRIEVE COMPUTER RESOURCE SPECIFIC COMMANDS
204
ATTACHED LABELS?
206
YES
RETRIEVE COMMANDS AND ATTACH AS CASCADE MENUS FOR LABEL NAMES
208
NO
RETRIEVE CONTAINER SPECIFIED COMMANDS
210
DISPLAY CONTEXT MENU
212
END
214

Fig. 5

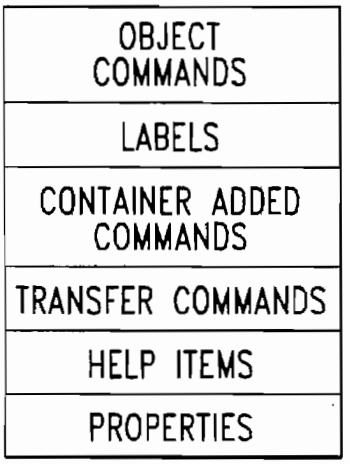

OBJECT COMMANDS
LABELS
CONTAINER ADDED COMMANDS
TRANSFER COMMANDS
HELP ITEMS
PROPERTIES

Fig. 6

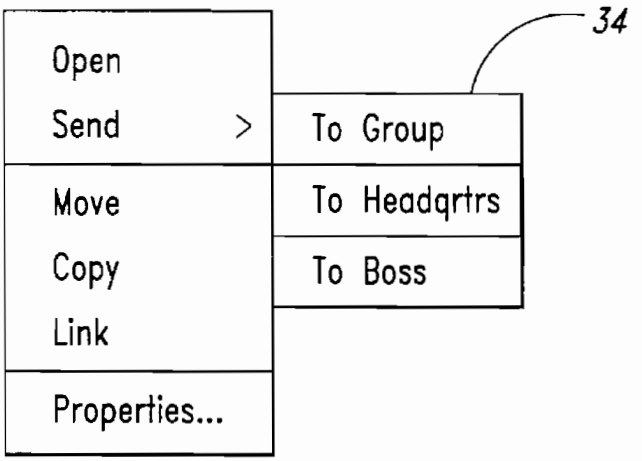

34
Open
Send >
Move
Copy
Link
Properties...
To Group
To Headqrtrs
To Boss

Fig. 8

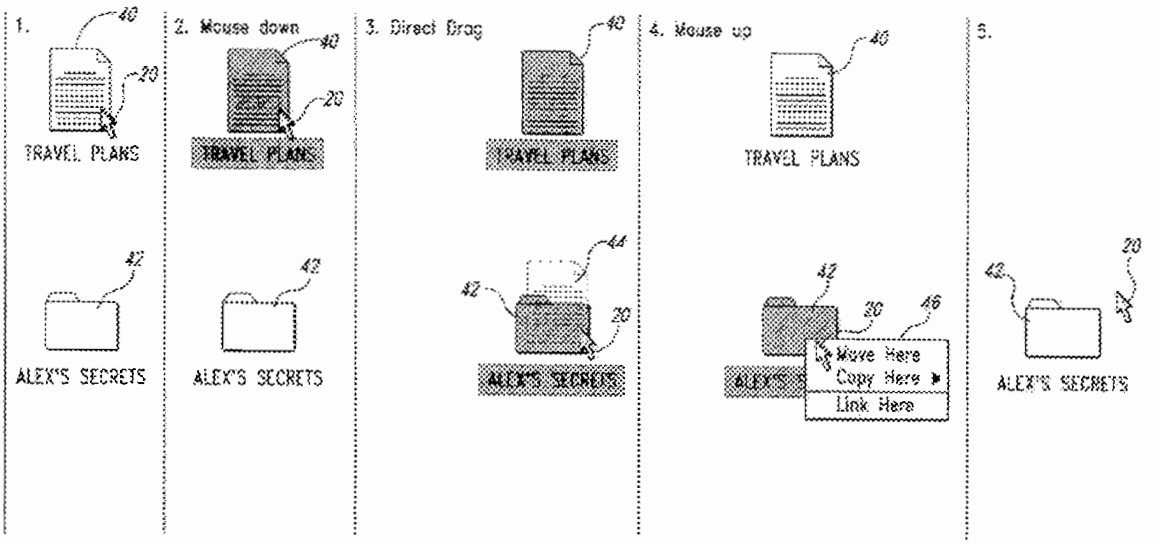

1.
2. Mouse down
3. Direct Drag
4. Mouse up
5.
40
20
TRAVEL PLANS
42
ALEX'S SECRETS
44
30
46
Move Here
Copy Here
Link Here

Fig. 7

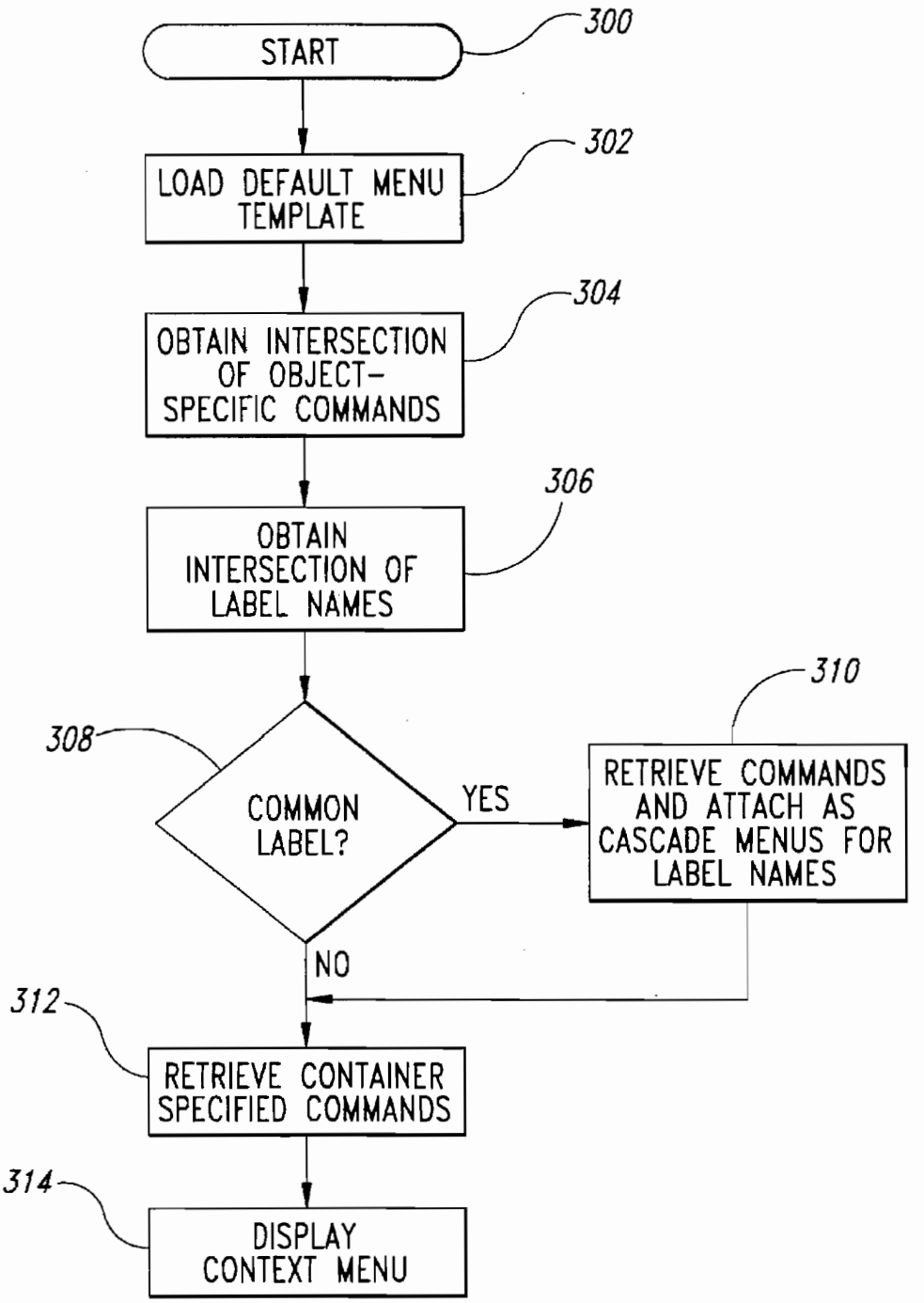

START
300
LOAD DEFAULT MENU TEMPLATE
302
OBTAIN INTERSECTION OF OBJECT-SPECIFIC COMMANDS
304
OBTAIN INTERSECTION OF LABEL NAMES
306
COMMON LABEL?
308
YES
RETRIEVE COMMANDS AND ATTACH AS CASCADE MENUS FOR LABEL NAMES
310
NO
RETRIEVE CONTAINER SPECIFIED COMMANDS
312
DISPLAY CONTEXT MENU
314

Fig. 9

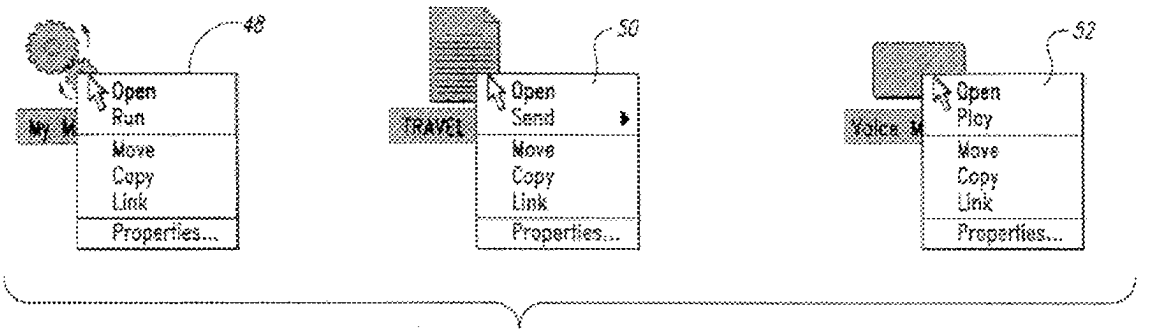


Fig. 10A

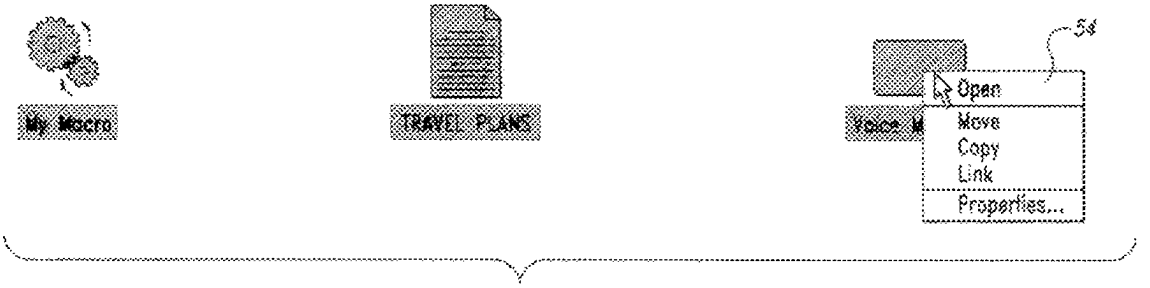


Fig. 10B

CONTEXT SENSITIVE MENU SYSTEM/ MENU BEHAVIOR

CROSS-REFERENCE TO RELATED APPLICATION

This application is a continuation of U.S. patent application Ser. No. 08/166,339, filed Dec. 13, 1993, now abandoned.

AREA OF THE INVENTION

This invention relates to the field of user interfaces for computer systems, and more particularly to graphical user interfaces wherein a user selects from a collection of graphical representations displayed upon a video screen corresponding to actual computer resources.

BACKGROUND OF THE INVENTION

The United States has experienced an extraordinary expansion in the ownership and utilization of computers. Computers, once considered primarily the tools of scientists, can now be found in a substantial portion of the homes and businesses across the country. Though partial credit for the unprecedented growth in the utilization of computers is attributable to lower costs associated with manufacturing computers and related peripheral devices which has made such equipment affordable to a much larger segment of the public, the credit is equally, if not more, attributable to the enhanced versatility, ease in learning to use, and ease of using computers which is provided by the operating systems and applications software running on the now affordable computers.

It is therefore very important when designing an operating system, and more particularly a user interface for a computer system, to provide a high degree of user friendliness, which incorporates ease of learning and ease in using the computer system. Users appreciate a system which enables them to accomplish their computer related tasks with the least amount of obstacles and delays. A considerable number of the computer users/operators today base their like or dislike of a computer system upon the user interface's time and effort saving features as well as display features which enhance the interface's aesthetic appeal.

The Windows operating system and its series of menus and buttons has simplified the use of the computer system. The Windows operating system is unquestionably easier to learn than earlier operating systems. Even more importantly the Windows operating system enables a user to access a great number of computer resources from any given screen by selecting ones of the many menu items and control buttons.

Another concern of computer interface developers is the learnability of new interface features. One of the believed advantages of the newer mouse-type menu driven operating systems is the ease with which users learn how to invoke the various system commands. Much of the success in improving the usability of the operating systems is attributed to the large amount of visual information provided at each decision making step.

A known manner for presenting the commands and resources available to a user is the use of menus. Menus have been presented to the user in the form of menu bars presenting selections appropriate for a given window or container, pop-up menus presenting choices appropriate for an object, and tear off menus which are characterized by their persistence on the display screen even after a user has made a selection.

Many applications have become so complex that the set of choices provided to the user for an application by means of the menu bars, which are generally intended to cover all types of objects in all possible contexts, becomes unmanageable when all of the choices are presented at the same time. It is therefore desirable for a computer system to provide some mechanism for restricting the set of choices presented to a user.

Furthermore, in view of the complex user interfaces existing today providing the user with a large number of choices and many diverse manners to access the choices, it is important to provide the user with a robust user interface selection system for guaranteeing that the computer system executes commands according to the desires of the user. To this end, safeguards against inadvertent selections are extremely desirable. In a graphical user interface, the user tends to rely heavily upon visual prompts and therefore safeguards are most effectively implemented when they incorporate some visual aspect into the safeguard.

A noticeable trend in graphic user interfaces has been to increase the amount of information displayed on a display screen. Menus have become less manageable. Multiple selection modes increase the likelihood of user confusion. It is therefore desirable to counter the increased cluttering of the display screen by the menus provided by the present graphical user interfaces, and the resulting user confusion, by reducing the number of choices presented on the user interface at any given time and by presenting improved visual prompts to inform the user of the current state of the user interface. It is desirable to maintain or even enhance the ease with which computer resources are accessed by means of the graphical user interface.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to display a reduced number of choices to a user which apply specifically to a selected computer resource or set of resources.

It is a further object of the present invention to present the reduced set of choices in a manner that clearly indicates to the user the state of the computer system at each state of the selection/execution process for a computer resource.

It is a further object of the present invention to present the choices provided to a user by a graphical user interface in a manner that the user may quickly and easily select/execute the desired computer resource.

The above and other objects are achieved by a computer system having a graphical user interface which presents a set of representations corresponding to computer resources including objects and controls. When a user selects a computer resource by placing the display pointer over a computer resource and clicking a context button, or by any other suitable selection signal to the computer system, the computer system registers the computer resource and displays a context menu when the user releases the context button. The context menu presents a set of choices to the user based primarily upon the selected computer resource. The set of choices is secondarily determined by the particular environment in which the computer resource resides at the time of the selection.

The manner in which the choices are presented and the behavior of the context menu, which is also applicable to the other known menu types, are summarized below. First, the computer system visually tracks the movement of the display pointer when the user releases the context button after selecting the computer resource and when the user depresses a button while the context menu is displayed. The computer

system provides two distinguishable highlight display features in order to provide visual feedback to the user of the present selection mode of the computer system.

Furthermore, menu of the present invention includes executable cascade items and non-executable cascade names. Both cascade items and cascade names are associated with cascade menus which the computer system displays after a short delay when the user positions the display pointer over the cascade item/name. Both cascade items and names include executable items in the resulting cascade menus. However, clicking a mouse while the display pointer is positioned over the cascade item causes the computer to execute the default operation listed in the cascade menu for the cascade item.

In addition, in order to indicate to the user that a cascade item is executable, the text of the cascade item is highlighted when the user positions the display pointer over the cascade item. The text of a cascade name, which is not executable, does not highlight when a display pointer is moved over a cascade name.

Clicking the context button while the display pointer is positioned on an active menu item causes the computer system to dismiss the context menu and perform the operation associated with the menu item. Suitable menu items include objects, commands and controls. If the menu item is an object, the computer system performs a default operation associated with the object.

The combination of the above described features into menu features/behavior of a graphical user interface provide a less cluttered, easy to learn, easy to use, intuitive, and robust system for users to access and control applications.

BRIEF DESCRIPTION OF THE DRAWINGS

The appended claims set forth the features of the present invention with particularity. The invention, together with its objects and advantages, may be best understood from the following detailed description taken in conjunction with the accompanying drawings of which:

FIG. **1** is a schematic drawing of an exemplary computer system incorporating the present invention;

FIG. **2** is a sequence of views of a display screen illustrating the relation between a mouse, a display pointer and a context menu for a selected computer resource;

FIG. **3** is a flow diagram summarizing the steps executed by a computer system in generating a context menu for a selected computer resource and making a selection from the generated context menu;

FIGS. **4***a* and **4***b* illustrate the differing context menus for a standard icon for a document and a mini (open) icon for the opened document;

FIG. **5** is a flow diagram summarizing the steps executed by the computer system to generate a context menu;

FIG. **6** is a schematic depiction of the fields of the context menu;

FIG. **7** is a series of views illustratively depicting the states of two related objects while generating a context menu based upon multiple related objects;

FIG. **8** is an illustrative example of a cascade name and its cascade menu;

FIG. **9** is a flow diagram summarizing the steps for generating a context menu for a set of selected objects; and

FIGS. **10***a* and **10***b* provide an illustrative example of an intersection of context menus for multiple selected objects.

DETAILED DESCRIPTION OF THE DRAWINGS

The computer system schematically illustrated in FIG. **1** comprises a central processing unit (CPU) **2** coupled by means of a bus **6** in a known manner to a random access memory **4**. The CPU **2** is also coupled to a non-volatile secondary memory **8** for storing various system and applications routines and programs. The CPU **2** is coupled in known manner to a user interface **10** including a video display **12**. The video display **12** may be any of a number of known display devices including, for example, monochrome and color cathode ray tubes and LCD displays. The user interface **10** also includes a keyboard **14** and mouse **16** to facilitate the submission of instructions to the CPU **2**. The mouse **16** includes a default selection button **17** and a context button **18**. Though not shown in FIG. **1**, the computer system may also include a number of peripheral units as would be known to those skilled in the art. The computer system hardware depicted in FIG. **1** is intended merely to show a representative hardware configuration. It would of course be understood by one of ordinary skill in the art that the present invention encompasses other computer system hardware configurations and is not limited to the computer system hardware configuration described above.

Turning now to FIG. **2**, a set five columns of icons are provided to demonstrate the various modes of a computer resource having a context menu. The steps for carrying out the generation and manipulation of a context menu for a computer resource are described below in conjunction with the flow diagram provided in FIG. **3** as well as other figures which expand upon certain steps depicted in FIG. **3**. In addition, FIG. **2** depicts a number of unique menu behaviors which enhance the utility of context menus.

The preferred mouse for the present invention includes the default selection button **17** which, in a known manner, selects a computer resource during a first click of the default selection button **17** and executes a default command upon the selected computer resource in response to clicking the default selection button **17** while the display pointer remains on the selected computer resource.

The preferred mouse for the present invention also includes the context button **18**. The operation of the context button **18** closely parallels the operation of the default selection button **17**. The operation of the context button **18** is summarized in FIG. **3**. As shown in column **1** of FIG. **2**, the user positions the display pointer **20** over a file icon **22**. In accordance with steps **100** and **102** of FIG. **3**, pressing down on the context button **18** while the display pointer **20** is displayed over the file icon **22** "selects" that computer resource. The computer system visually indicates to the user that the computer resource corresponding to the file icon **22** has been selected by highlighting the file icon **22** in a known manner. For backgrounds having an associated context menu, the computer system also generates an insertion point **24** resembling an asterisk. Th insertion point **24** provides the user with a point of reference for where the computer system will display a context menu. Thus, the behavior of the context menu portion of the computer system provides the user with some degree of flexibility in deciding where the context menu for an object will be displayed. Column **2** of FIG. **2** depicts a highlighted selected file icon **26**. Other suitable means for highlighting a selected computer resource would be known to those of ordinary skill in the art of graphic user interfaces. Control then passes to step **104** wherein the computer system waits for the user to release the context button **18**.

When the context button **18** is released, control passes to step **106** wherein the computer system determines whether the display pointer **20** is still over the computer resource previously selected when the user depressed the context button **18**. If the user releases the context button **18** while the

display pointer **20** is displayed on the selected computer resource, control passes to step **108** wherein the computer system generates a context menu for the selected computer resource. In the preferred embodiment of the present invention, the computer system generates context menus for a variety of computer resources including: system level objects, document sub-parts, controls, and title bars. At the completion of step **108**, the computer system displays a context menu for the computer resource comprising a list of selections.

In contrast to known menu bars, the items listed on the context menu are determined primarily by the computer resource for which they are displayed instead of a surrounding container. Once the initial set of menu items is generated based upon the selected computer resource(s), the computer system augments the set of menu items specific to the computer resource(s) with additional menu items based upon the context in which a user selects the computer resource(s). For example, turning to FIGS. **4***a* and **4***b*, the context menu for the icon **28** for a closed document shown in FIG. **4***a* does not include the "close" menu item. However, turning to FIG. **4***b*, the icon **30** for the opened document includes the "close" menu item, but does not include the "open" menu item. Another example of context is a context menu for an object embedded within a document differing from the same object embedded within a folder.

The steps for generating a context menu are summarized in FIG. **5** and are described below. Upon completion of these steps, the selections listed upon context menus include one or more of the following: commands specific to the object such as Edit, View, Open, and Print; commands added by the container such as Delete; selection specific commands such as formatting commands for a document sub-part; control commands specific to a particular control; source and destination transfer commands; and properties.

Turning now to FIG. **5** the steps are summarized for the generation of a context menu for a computer resource. From the start **200** of the context menu generation procedure, control passes to step **202** wherein the computer system loads the default menu template into the list of items and names for the context menu. The default menu template, provided for all context menus regardless of the container or computer resource for which the context menu is being created, includes a known list of transfer related commands, help items and properties. Control then passes to step **204**.

At step **204**, the computer system retrieves a set of commands relating to the specific class of the computer resource for which the context menu is being generated and adds the set of object specific commands to the list of items and names for the context menu. Control then passes to step **206** wherein the computer system determines whether any labels are attached to the computer resource. An example of a label in a computer resource is a "cc:" label in a document. If the computer resource contains attached labels, then control passes to step **208** wherein the computer system retrieves all commands relating to the attached labels. Each attached label name (for which a command is retrieved) is added to the context menu list for the resource, and the commands relating to each label are attached to the label name as cascade menu items. Control passes next to step **210**, wherein the computer system retrieves commands provided by the container in which the computer resource resides and adds the container specific commands to the context menu list.

If the computer system determines at step **206** that no labels are attached to the computer resource, then control passes to step **210** described above. Control then passes to step **212** wherein the computer system in a known manner transforms the list of context menu commands compiled during the preceding steps into a displayed context menu. The structure of the fields of the context menu are depicted in FIG. **6**. Thereafter, control passes to the end step **214** and control passes to step **112** of the procedure summarized in FIG. **3**.

Returning to FIG. **2**, the file icon **32** and context menu **34** at the head of column **3** of FIG. **2** depict the initial appearance of a context menu generated for a selected file icon. As previously mentioned, examples of other computer resources for which the computer system of the present invention generates context menus include system level objects, document sub-parts, controls, and title bars. The computer system highlights a default item for the context menu. The default item is the selection that the computer system executes automatically when the user selects and executes an operation on an object by means of the default selection button **17**. In this particular example, the default item is the "Open" item which is highlighted in bold typeface.

At step **112**, the computer system commences enhanced visual tracking of the position of the display pointer **20** upon the context menu and waits for receipt of a signal indicating that the user depressed either the context button **18** or default selection button **17**. Continuing with the description of FIG. **2**, column **3** includes a series of 3 views which depict tracking/display features of a context menu that enhance the utility of the graphic user interface. In the example provided in column **3** of FIG. **2**, the computer system tracks movement of the display pointer **20** by changing the color or shade of the text of an active menu item contained in the menu when the display pointer **20** points to the active menu item. In the middle view of column **3** of FIG. **2**, the computer system highlights the text for the "Link" menu item since the display pointer **20** is within the boundaries of the "Link" menu item. As illustrated by the bottom view of column **3** of FIG. **2**, if the display pointer **20** is moved outside the context menu **34** during step **112**, no text is highlighted.

A context menu may also include cascade names and cascade items. A cascade name is not executable. Instead a cascade name provides a menu including a set of executable items. The user chooses an executable menu item from a cascade menu corresponding to a cascade name, and the computer system then performs the operation corresponding to the cascade menu item. Since cascade names are not executable, the text of a cascade name is not highlighted (as described above for executable menu items) during menu tracking.

If the user moves the display pointer **20** over a cascade menu name in the context menu **34** such as the "Send" selection in the example in FIG. **2**, then only the triangle **36** changes its color or shade and a cascade menu (illustratively depicted in FIG. **8**) automatically appears after a short delay to prevent flashing of the cascade when the display pointer **20** is merely drawn quickly through the cascade name in order to access another menu selection.

Cascade items, in addition to providing a cascade menu having executable menu items, are executable. Execution of the cascade item without selecting a menu item within the cascade menu causes the computer system to select a default menu item from the cascade menu.

Another feature of cascade menus for the context menus is the ability to move the display pointer **20** outside the cascade name area (in the illustrative example, the "Send"

menu selection area) or the area for the cascade menu corresponding to the cascade name for a short period of time without causing the computer system to retract the cascade menu. Furthermore, if the user drags the display pointer **20** over another context menu selection such as the "Move" selection in order to access the items in a cascade menu (not shown) associated with the "Send" cascade name, the computer system will not respond to the presence of the display pointer **20** within the "Move" selection area. Incorporating this known method for accessing a cascade menu into the specific environment of a context menu further enhances the utility of the context menu by enabling a user to access the items of a cascade menu by moving the display pointer **20** in a diagonal direction instead selecting a cascade menu item by means of a horizontal movement of the display pointer **20** onto the cascade menu and then a vertical movement to a specific cascade menu item.

The display characteristics for cascade menu items follow the same behavior as the menu items presented on a context menu. The computer system highlights an active cascade menu item when the user positions the display pointer **20** within the boundaries of the active cascade menu item, and cascade menu items are executed in the same manner as context menu items.

Continuing with the description of FIG. **3**, when either the default selection button **17** or the context button **18** is depressed by the user at step **112**, control passes to step **114**. If the computer system determines in a known manner that the display pointer **20** was outside the context menu **34** and the user depressed the default selection button **17** or the context button **18**, then control passes to step **116** and the computer system dismisses the displayed context menu **34**. Control then passes to step **118** wherein if the context button **18** was depressed to cause the dismissal of the context menu **34** during step **116** then control passes to step **102**. If however the default selection button **17** was depressed to cause the dismissal of the context menu **34** during step **116**, then control passes to the end step **120**.

Continuing with the description of step **114**, if the user depresses the context button **18** or the default selection button **17** while the display pointer **20** is within the context menu **34**, then control passes to step **122** wherein the computer system continues to track the movement of the display pointer **20** and waits for the user to release the button. The views provided in column **4** of FIG. **2** illustratively depict the behavior of the context menu **34** at step **122**. The tracking behavior of the context menu after the user depresses the default selection button **17** or the context button **18** is equivalent to the tracking behavior described in association with column **3** of FIG. **2**; however, the background for an active menu item is highlighted rather than the text of the active menu item. This feature is depicted by the background **38** for the "Open" selection in the top view of column **4** in FIG. **2**. The highlighted background **38** for the selection indicates to the user that the highlighted selection will be executed when the user releases the button.

When the user releases the default selection button **17** or the context button **18** at step **122**, control passes to step **124**. If the display pointer **20** is outside the context menu **34** (at step **124**), then control passes to step **126** wherein the computer system dismisses the context menu **34**. Control then passes to the end step **120**.

If however at step **124** the display pointer **20** is within the context menu **34**, then control passes to step **128**. If at step **128** the computer system determines that the display pointer **20** is over an active menu item such as the "Open" menu item in FIG. **2**, then control passes to step **130** wherein the context menu **34** is dismissed. Control then passes to step **132** wherein the computer system performs the "Open" operation on the file. Control then passes to the end step **120**.

If at step **128** the display pointer is not over an active menu item, then control passes to step **134**. If at step **134** the computer determines that the display pointer **20** is over an active cascade item, then control passes to step **136** wherein the context menu is dismissed. Control then passes to step **138** wherein the computer system identifies a default cascade menu item. The computer system then performs the operation associated with the default cascade menu item. If the user selects a cascade item at step **122**, then control of the computer system will eventually pass to step **138** wherein the computer system executes the default cascade menu item for the cascade item.

If at step **134** the computer determines that the context button **18** was released while the display pointer **20** was positioned over an inactive menu selection or the pointer **20** was positioned over a cascade name, then control passes to step **112** and the computer system continues to display the context menu **34** and track the position of the display pointer **20** as depicted in column **3** of FIG. **2**.

If the computer system at step **106** determines that a user selected a first computer resource and then moved the display pointer **20** outside the display region for a first computer resource before releasing the context button **18**, then no context menu will appear for the first computer resource. Instead, control passes to step **110** wherein the computer system generates a context menu for a second computer resource pointed to by the display pointer **20** when the user released the context button **18**.

Turning briefly to FIG. **7**, the operation of dropping a document into a folder using the context button **18** is depicted by a series of five display states for a file icon **40** and a folder icon **42**. In the first state, the user has moved the display pointer **20** over a file icon **40**. In the second state, the user has selected the file icon **40** by pressing down upon the context button **18**. This corresponds to step **102** in FIG. **2**. In the third state, the user has moved the display pointer **20** over a folder icon **42**. As the user moves the display pointer **20** while holding the context button **18** down, the computer system displays a ghost icon **44** corresponding to the file icon **40** to visually inform the user that the file icon **40** was selected before moving the display pointer **20**.

In the forth state, the user has released the context button **18**, and the computer system, in accordance with step **110** of FIG. **5** has generated a context menu **46** listing operations that can be executed by the computer system based upon the relationship between the file icon **40** and folder icon **42**.

The default operation, presented in bold typeface, is the "Move Here" operation. Execution of the "Move Here" menu item in accordance with the steps depicted in FIG. **2** and previously described above causes the file corresponding o the file icon **40** to be placed in the folder corresponding to the folder icon **44**. The fifth state depicted in FIG. **7** shows the state of the display after the default command, "Move Here", has been executed by the computer system. As is apparent from the view of the fifth state, the file icon **40** disappears from the screen to reflect the placement of the file into the folder corresponding to the folder icon **44**.

It should be noted that even though the generation of a context menu for a computer resource in accordance with the present invention has been described with respect to a user manipulating a mouse controlled display pointer **20**, equivalent operation of the context menu is provided

through the use of pre-arranged keyboard signals with equivalent display behavior by the graphic user interface. In some cases, the user may actually use both the keyboard **14** and mouse **16** to generate a context menu and execute an item on the context menu.

The computer system is capable of generating a context menu for multiple selected objects. If the multiple selected objects contain differing context menus, then the context menu generated for the multiple selected objects is the intersection of the menu items for each of the selected objects.

The steps of the procedure for obtaining a context menu for multiple selected objects having different context menus are summarized in FIG. **9**. When the user releases the context button **18** after selecting multiple objects, control passes to the start **300** of the procedure summarized in FIG. **9**. Control then passes to step **302** wherein the computer system loads the default menu template into the list of items and names for the context menu in the same manner described for a context menu for a single context menu. Control then passes to step **304** wherein the computer system retrieves a separate list of object specific commands for each of the selected objects, then compares the lists to obtain the intersection of the lists of object specific commands for the selected objects. The computer system adds any object specific commands common to all of the selected objects to the items and names for the context menu list. Control then passes to step **306** wherein the computer system obtains the intersection of the labels for the selected resources. Control then passes to step **308**.

If the computer system determines at step **308** that there are labels common to all of the objects, then control passes to step **310** wherein the computer system retrieves all commands relating to the labels. Each label name for which commands were retrieved is added to the context menu list, and the retrieved commands are added as cascade menus for the label names added to the context menu list. Control then passes to step **312**.

If at step **308** the computer system determines that there are no labels common to all of the selected objects then control passes to step **312** wherein the computer system retrieves commands provided by the container in which the objects reside and adds the container specific commands to the context menu list. Control then passes to step **314** wherein the computer system in a known manner transforms the list of context menu commands compiled during the preceding steps into a displayed context menu for the selected group of objects. Thereafter, control passes to the end step **316**.

Turning to FIG. **10***a*, a context menu for a first object **48** contains a "Run" menu item, a context menu for a second object **50** contains a "Send" cascade name, and a context menu for a third object **52** contains a "Play" menu item in its context menu. Since the "Run", "Send", and "Play" menu items/name are not contained in all three context menus, the context menu **54** in FIG. **10***b* resulting at the end of step **108** from the multiple selection of all three of the objects does not include the "Run", "Send", or "Play" menu items/name. If the user selects the multiple objects and then clicks the default selection button **17**, then the computer system executes the default action for each of the computer resources.

A further enhancement to the behavior of menus is the inclusion of file system objects as menu items. Selecting an object in a menu causes the computer system to execute the default command on the object.

A preferred embodiment Of a system for providing context menus in a graphical user interface environment has been described. It would of course be known to one of ordinary skill in the area of user interfaces for computers and operating system in general to make certain modifications and to the afore-described methods and system which would not depart from the scope and spirit of the invention described in the claims appended hereinafter. In particular, though a preferred method for carrying out context menus has been presented, it will be recognized by those skilled in the art that the above invention can be carried out in a number of different manners including rearranging or even substituting certain ones of the steps summarized in the flow diagrams as well as using alternative appropriate data structures relating to the context menus, and to consult information of a different type than that specifically listed in order to determine the context in which a menu for a selected computer resource is generated.

It is contemplated that even though use of a keyboard to select computer resources in accordance with the steps of the present invention is no considered the best way to implement the present invention, a keyboard can replace the mouse controlled display pointer without departing from the scope of the present invention.

Furthermore, even though new menu behavior has been described using a context menu as a primary example, the described menu behavior applies as well to other menus including menu bars and tear off menus. Tear off menus are a known menu type. A tear off menu behaves substantially the same as a context menu. However, a tear off menu is always active, and a tear off menu is dismissed by executing a close command upon the tear off menu.

It would also be known to utilize the present invention within other computer configurations such as a local area network, or a group of computer work stations sharing a mainframe operating system.

What is claimed is:

1. In a computer system having a central processing unit (CPU), a graphical user interface including a display and a user interface selection device communicatively coupled to the CPU, a method for providing, and selecting from, a menu for a selected computer resource, said method comprising the steps of:

generating a set of menu selections for the selected computer resource in response to receiving, by the CPU, a context menu generation signal from the user interface selection device, the generating step comprising the steps of:

retrieving a menu selection relating to a class of objects to which the selected computer resource belongs; and

retrieving a menu selection associated with a container in which the selected computer resource resides; and

displaying upon the display the set of menu selections in a menu positioned in the proximity of a graphical representation of the selected computer resource.

2. The method of claim **1** wherein the step of generating a set of menu selections further comprises the step of:

retrieving a label based menu selection based upon a label contained within the selected computer resource.

3. The method of claim **2** wherein the step of generating a set of menu selections further comprises the step of retrieving a label name menu selection for the label contained within the selected computer resource, and wherein the displaying step further comprises displaying a cascade menu adjacently to the label name menu selection, the cascade menu including the label based menu selection.

4. The method of claim 1, wherein the user interface selection device includes a default execution button and a separate and distinct context button, and wherein the context menu generation signal comprises a context button up signal transmitted by the user interface selection device.

5. The method of claim 1 further comprising the steps of:

visually tracking, in a first mode, the positioning of a display pointer within the set of displayed menu selections; and

receiving, by the CPU, a first signal while visually tracking in the first mode and, in response thereto, visually tracking, in a second mode visually distinguishable from the first mode, the positioning of the display pointer within the set of displayed menu selections.

6. The method of claim 5 further comprising the steps of:

receiving, by the CPU, a second signal while visually tracking in the second mode and, in response thereto, performing the steps of:

identifying a one of the set of displayed menu selections on which the display pointer is positioned; and

performing an operation associated with the one of the set of displayed menu selections.

7. The method of claim 6 wherein the first signal comprises a button down signal transmitted by a mouse, and the second signal comprises a button up signal transmitted by the mouse.

8. The method of claim 1 wherein the set of displayed menu selections includes a file system object menu selection corresponding to a file system object, and wherein said method further comprises the steps of receiving, by the CPU, an execution signal while a display pointer is positioned on the file system object menu selection and, in response thereto, performing a default command associated with the file system object menu selection.

9. In a computer system having a central processing unit (CPU), a graphical user interface including a display and a user interface selection device communicatively coupled to the CPU, a method for providing, and selecting from a menu associated with a second selected computer resource, said method comprising the steps of:

selecting a first computer resource in response to receiving, by the CPU, a first signal from the user interface selection device;

generating a set of menu selections for a second computer resource associated with a transfer of the first computer resource to the second computer resource in response to receiving, by the CPU, a second signal from the user interface selection device after selecting the first computer resource; and

displaying upon the display the set of menu selections in a menu positioned in the proximity of a graphical representation of the second computer resource.

10. The method of claim 9 wherein the first computer resource is a document and the second computer resource is a folder.

11. The method of claim 9 wherein the user interface selection device includes a default execution button and a separate and distinct context button, further comprising the steps of:

transmitting by the user interface selection device a context button down signal comprising the first signal; and

transmitting by the user interface selection device a context button up signal comprising the second signal.

12. In a computer system having a central processing unit (CPU), a graphical user interface including a display and a user interface selection device communicatively coupled to the CPU, a method for providing, and selecting from, a menu for a set of selected computer resources said method comprising the steps of:

generating a set of menu selections for the set of selected computer resources in response to receiving, by the CPU, a context menu generation signal and a multiple select mode signal from the user interface selection device, the generating step comprising the steps of:

retrieving a menu selection obtained by taking the intersection of sets of object specific commands associated with the set of selected computer resources; and

retrieving a menu selection associated with a container in which the set of selected computer resources resides; and

displaying upon the display the set of menu selections in a menu.

13. The method of claim 12 wherein the step of generating a set of menu selections further comprises the step of:

retrieving a label based menu selection based upon a label contained within each computer resource of the set of selected computer resources.

14. The method of claim 13 wherein the step of generating a set of menu selections further comprises the step of retrieving a label name menu selection for the label contained within each selected computer resource, and wherein the displaying step further comprises displaying a cascade menu adjacently to the label name menu selection, the cascade menu including the label based menu selection.

15. The method of claim 12, wherein the context menu generation signal comprises a context button up signal transmitted by a mouse.

16. The method of claim 12 further comprising the steps of:

visually tracking, in a first mode, the positioning of a display pointer within the set of displayed menu selections; and

receiving, by the CPU, a first signal while visually tracking in the first mode and, in response thereto, visually tracking, in a second mode visually distinguishable from the first mode, the positioning of the display pointer within the set of displayed menu selections.

17. The method of claim 16 further comprising the steps of:

receiving, by the CPU, a second signal while visually tracking in the second mode and, in response thereto, performing the steps of:

identifying a one of the set of displayed menu selections on which the display pointer is positioned; and

performing an operation corresponding to the one of the set of displayed menu selections.

18. A graphical user interface providing context sensitive menu options to a user for a selected computer resource in a computer system having a central processing unit (CPU), a display, and a user interface selection device communicatively coupled to the CPU, the graphical user interface comprising:

means for generating a set of menu selections for the selected computer resource in response to receiving, by the CPU, a context menu generation signal from the user interface selection device, the means for generating comprising:

means for retrieving a menu selection relating to a class of computer objects to which the computer resource belongs, and

means for retrieving a menu selection associated with a container in which the computer resource resides; and

means for displaying upon the display the set of menu selections in a menu positioned in the proximity of a graphical representation of the selected computer resource.

19. The graphical user interface of claim **18** wherein the means for generating a set of menu selections further comprises means for retrieving a label based menu selection based upon a label contained within the selected computer resource.

20. The graphical user interface of claim **19** wherein the means for generating a set of menu selections further comprises means for retrieving a label name menu selection for the label contained within the selected computer resource, and wherein the means for displaying further comprises means for displaying a cascade menu adjacently to the label name menu selection, the cascade menu including the label based menu selection.

21. The graphical user interface of claim **18**, wherein the user interface selection device includes a default execution button and a separate and distinct context button, and wherein the context menu generation signal comprises a context button up signal transmitted by the user interface selection device.

22. The graphical user interface of claim **18** further comprising:

a first visual tracking mode for highlighting on the display in a first manner the positioning of a display pointer within the set of displayed menu selections;

a second visual tracking mode for highlighting on the display in a second manner, visually distinguishable from the first manner, the positioning of the display pointer within the set of displayed menu selections; and

means for switching from the first visual tracking mode to the second visual tracking mode in response to receiving, by the CPU, a first signal.

23. The graphical user interface of claim **22** further comprising:

selection execution means for identifying a one of the set of displayed menu selections on which the display pointer is positioned and performing an operation corresponding to the one of the set of displayed menu selections in response to receiving, by the CPU, a second signal while the second visual tracking mode is actively tracking the positioning of the display pointer.

24. The graphical user interface of claim **23** wherein the first signal comprises a button down signal transmitted by a mouse, and the second signal comprises a button up signal transmitted by the mouse.

25. The graphical user interface of claim **18** wherein the set of displayed menu selections includes a file system object menu selection corresponding to a file system object, and wherein the graphical user interface further comprises means for executing a default command associated with the file system object when the CPU receives an execution signal while a display pointer is positioned on the file system object menu selection.

26. A graphical user interface for providing, and selecting from a menu associated with a second selected computer resource in a computer system having a central processing unit (CPU), a display and a user interface selection device communicatively coupled to the CPU, said graphical user interface comprising:

means for selecting a first computer resource in response to receiving, by the CPU, a first signal from the user interface selection device;

means for generating a set of menu selections for a second computer resource associated with a transfer of the first computer resource to the second computer resource in response to receiving, by the CPU, a second signal from the user interface selection device after selecting the first computer resource; and

means for displaying upon the display the set of menu selections in a menu positioned in the proximity of a graphical representation of the second computer resource.

27. The graphical user interface of claim **26** wherein the first computer resource is a document and the second computer resource is a folder.

28. The graphical user interface of claim **26** wherein the user interface selection device includes a default execution button and a separate and distinct context button, and wherein the first signal comprises a context button down signal transmitted by the user interface selection device, and the second signal comprises a context button up signal transmitted by the user interface selection device.

29. A graphical user interface for providing, and selecting from, a menu for a set of selected computer resources in a computer system having a central processing unit (CPU), a display and a user interface selection device communicatively coupled to the CPU, said graphical user interface comprising:

means for generating a set of menu selections for the set of selected computer resources in response to receiving, by the CPU, a context menu generation signal and a multiple select mode signal from the user interface selection device, the means for generating comprising:

means for retrieving a menu selection obtained by taking the intersection of sets of object specific commands associated with the set of selected computer resources, and

means for retrieving a menu selection associated with a container in which the set of selected computer resources resides; and

means for displaying upon the display the set of menu selections in a menu.

30. The graphical user interface of claim **29** wherein the means for generating a set of menu selections further comprises means for retrieving a label based menu selection based upon a label contained within each computer resource of the set of selected computer resources.

31. The graphical user interface of claim **30** wherein the means for generating a set of menu selections further comprises means for retrieving a label name menu selection for the label contained within each selected computer resource, and wherein the means for displaying further comprises means for displaying a cascade menu adjacently to the label name menu selection, the cascade menu including the label based menu selection.

32. The graphical user interface of claim **29** wherein the context menu generation signal comprises a context button up signal transmitted by a mouse.

33. The graphical user interface of claim **29** further comprising:

a first visual tracking mode for highlighting on the display in a first manner the positioning of a display pointer within the set of displayed menu selections;

a second visual tracking mode for highlighting on the display in a second manner, visually distinguishable from the first manner, the positioning of the display pointer within the set of displayed menu selections; and

means for switching from the first visual tracking mode to the second visual tracking mode in response to receiving, by the CPU, a first signal.

34. The graphical user interface of claim **33** further comprising:

selection execution means for identifying a one of the set of displayed menu selections on which the display pointer is positioned and performing an operation corresponding to the one of the set of displayed menu selections in response to receiving, by the CPU, a second signal while the second visual tracking mode is actively tracking the positioning of the display pointer.

35. A computer-readable storage medium for use in a computer system having a display device, a selected object having a visual representation stored in storage, and a container object in which the selected object is contained, said medium holding instructions for:

adding a menu selection, related to the class of objects to which the selected object belongs, to a menu;

adding a menu selection that is associated with the container in which the selected object is stored to the menu; and

displaying the menu with the menu selections on the display device in proximity to the visual representation of the selected object.

36. The computer-readable storage medium of claim **35** wherein the selected object is a document.

37. The computer-readable storage medium of claim **35** wherein the container object is a folder.

38. A computer-readable storage medium for use in a computer system having a display device and a selected set of computer resources and a container object in which the selected set of computer resources resides stored in a storage device, the medium holding instructions for:

calculating an intersection of object-specific commands for the selected set of objects;

adding menu items for the calculated intersection of object-specific commands to a menu;

adding a menu item that is associated with the container to the menu; and

displaying the menu with the menu items on the display device.

* * * * *

US006578054B1

(12) **United States Patent**
Hopmann et al.

(10) **Patent No.: US 6,578,054 B1**
(45) **Date of Patent: *Jun. 10, 2003**

(54) **METHOD AND SYSTEM FOR SUPPORTING OFF-LINE MODE OF OPERATION AND SYNCHRONIZATION USING RESOURCE STATE INFORMATION**

(75) Inventors: **Alexander I. Hopmann**, Seattle, WA (US); **Rebecca L. Anderson**, Redmond, WA (US); **Brian J. Deen**, North Bend, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/412,766**

(22) Filed: **Oct. 4, 1999**

(51) **Int. Cl.**[7] **G06F 12/00**
(52) **U.S. Cl.** **707/201**; 707/203; 709/203
(58) **Field of Search** 707/201, 202, 707/203, 10, 8; 709/205, 201, 203, 204, 217; 711/161, 141, 144

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,600,834 A		2/1997	Howard	395/617
5,737,601 A		4/1998	Jain et al.	395/617
5,787,262 A	*	7/1998	Shakib et al.	707/201
5,812,773 A		9/1998	Norin	395/200.34
5,812,793 A		9/1998	Shakib et al.	395/200.31
5,884,325 A	*	3/1999	Bauer et al.	705/40
5,884,328 A	*	3/1999	Mosher, Jr.	707/202
5,924,094 A	*	7/1999	Sutter	707/1
5,924,096 A	*	7/1999	Draper et al.	707/10
5,991,771 A	*	11/1999	Falls et al.	707/201
6,058,401 A	*	5/2000	Stamos et al.	707/10
6,405,218 B1	*	6/2002	Boothby	707/201

OTHER PUBLICATIONS

Martin, J., "Design and Strategy for Distributed Data Processing", Prentice–Hall, 1998, pp. 272–304.*
Borenstein, et al., RFC 1521, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies," Sep. 1993.
Fielding, et al., RFC 2068, "Hypertext Transfer Protocol—HTTP/1.1," Jan. 1997.
Slein, et al., RFC 2291, "Requirements for a Distributed Authoring and Versioning Protocol For the World Wide Web," Feb. 1998.
Goland, et al., RFC 2518, "HTTP Extensions for Distributed Authoring—WEBDAV," Feb. 1999.
Fielding, et al., RFC 2316, "Hypertext Transfer Protocol—HTTP/1.1," Jun. 1999.
Yavin, D., "Replication's Fast Track,"BYTE, Aug. 1995, pp. 88a–88d, 90.

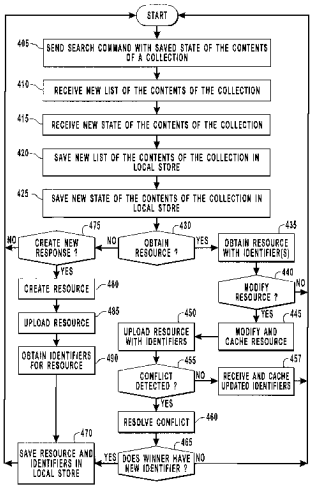
* cited by examiner

Primary Examiner—Meng-Al T. An
Assistant Examiner—Kenny Lin
(74) *Attorney, Agent, or Firm*—Workman, Nydegger & Seeley

(57) **ABSTRACT**

Systems and methods for synchronizing multiple copies of data in a network environment that includes servers and clients so that incremental changes made to one copy of the data can be identified, transferred, and incorporated into all other copies of the data. The synchronization can be accomplished regardless of whether modifications to the data have been made by a client while the client is in an on-line or off-line mode of operation. The clients cache data locally as data are modified and downloaded. The caching enables the clients to access the data and allows the synchronization so be performed without transmitting a particular version more than once between a client and a server. Such elimination of redundant data transmission results in an efficient use of time and network bandwidth.

20 Claims, 8 Drawing Sheets



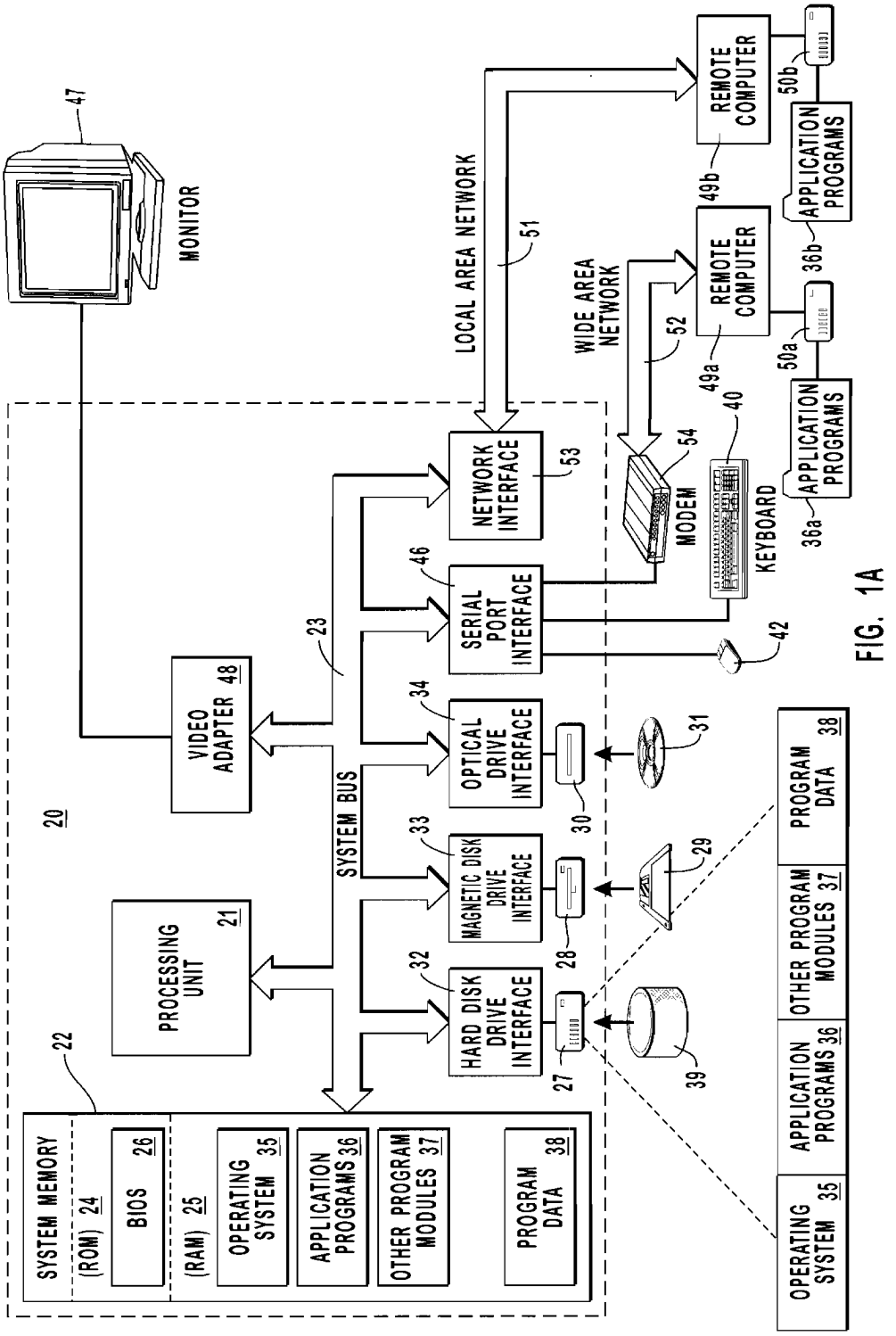

MONITOR
47
LOCAL AREA NETWORK
51
WIDE AREA NETWORK
52
REMOTE COMPUTER
49b
REMOTE COMPUTER
49a
50b
50a
APPLICATION PROGRAMS
36b
APPLICATION PROGRAMS
36a
NETWORK INTERFACE
53
MODEM
54
KEYBOARD
40
SERIAL PORT INTERFACE
46
42
VIDEO ADAPTER 48
23
OPTICAL DRIVE INTERFACE
34
31
30
MAGNETIC DISK DRIVE INTERFACE
33
29
28
HARD DISK DRIVE INTERFACE
32
27
39
20
PROCESSING UNIT
21
SYSTEM BUS
22
SYSTEM MEMORY
(ROM) 24
BIOS 26
(RAM) 25
OPERATING SYSTEM 35
APPLICATION PROGRAMS 36
OTHER PROGRAM MODULES 37
PROGRAM DATA 38
PROGRAM DATA 38
OTHER PROGRAM MODULES 37
APPLICATION PROGRAMS 36
OPERATING SYSTEM 35

FIG. 1A

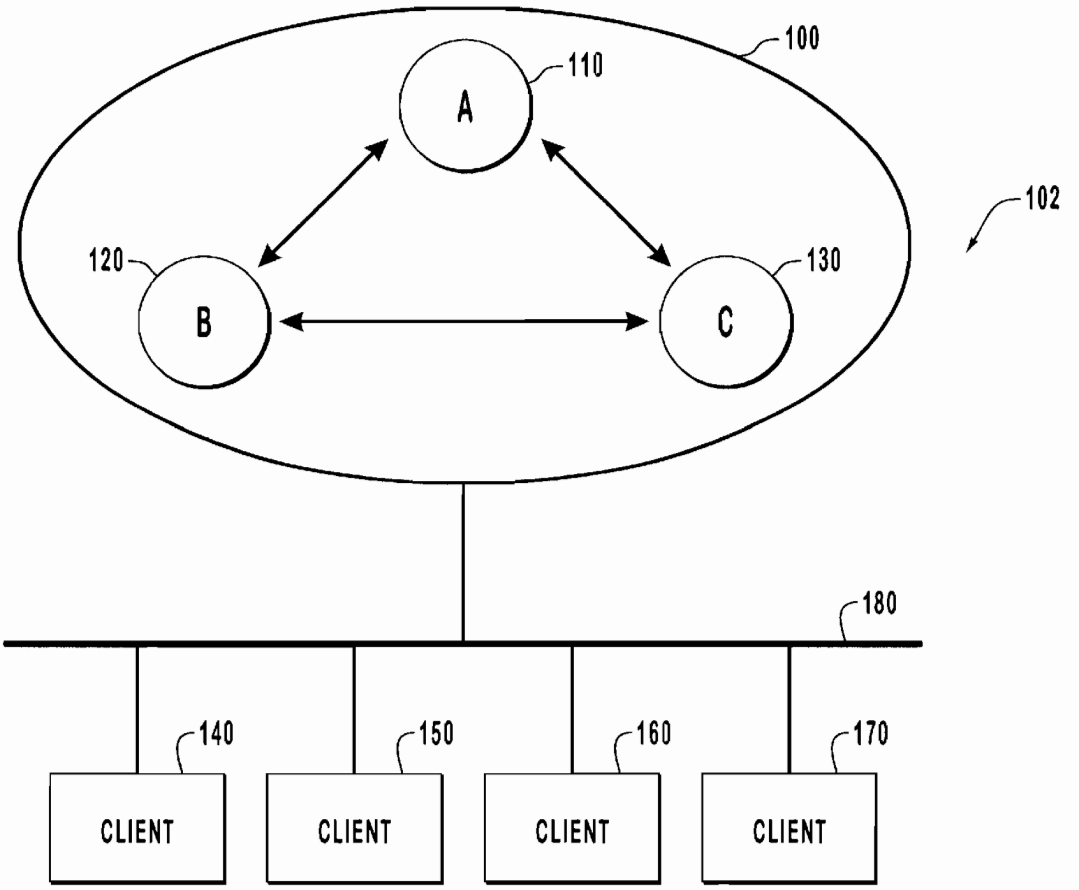

100
110
A
102
120
B
130
C
180
140
150
160
170
CLIENT
CLIENT
CLIENT
CLIENT

FIG. 1B

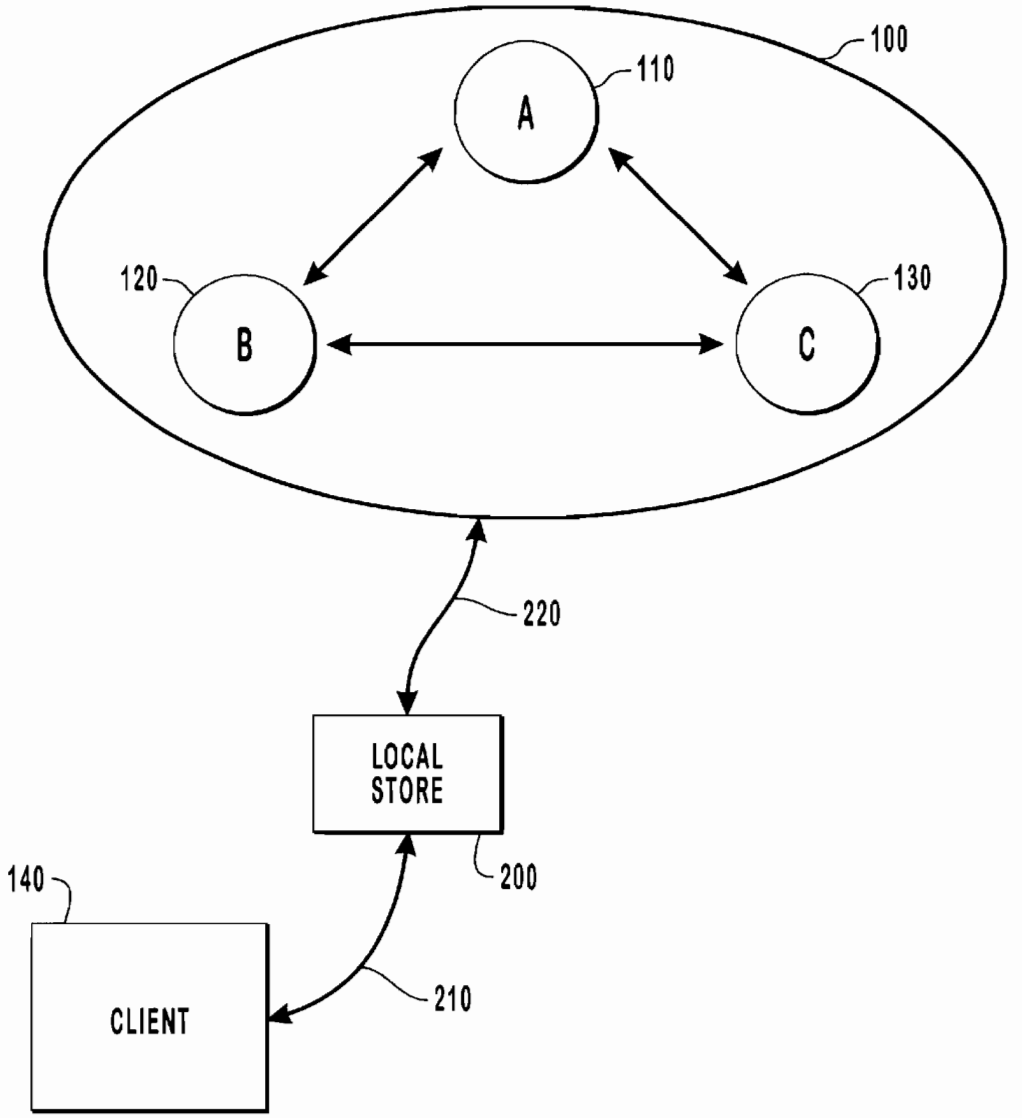


FIG. 2

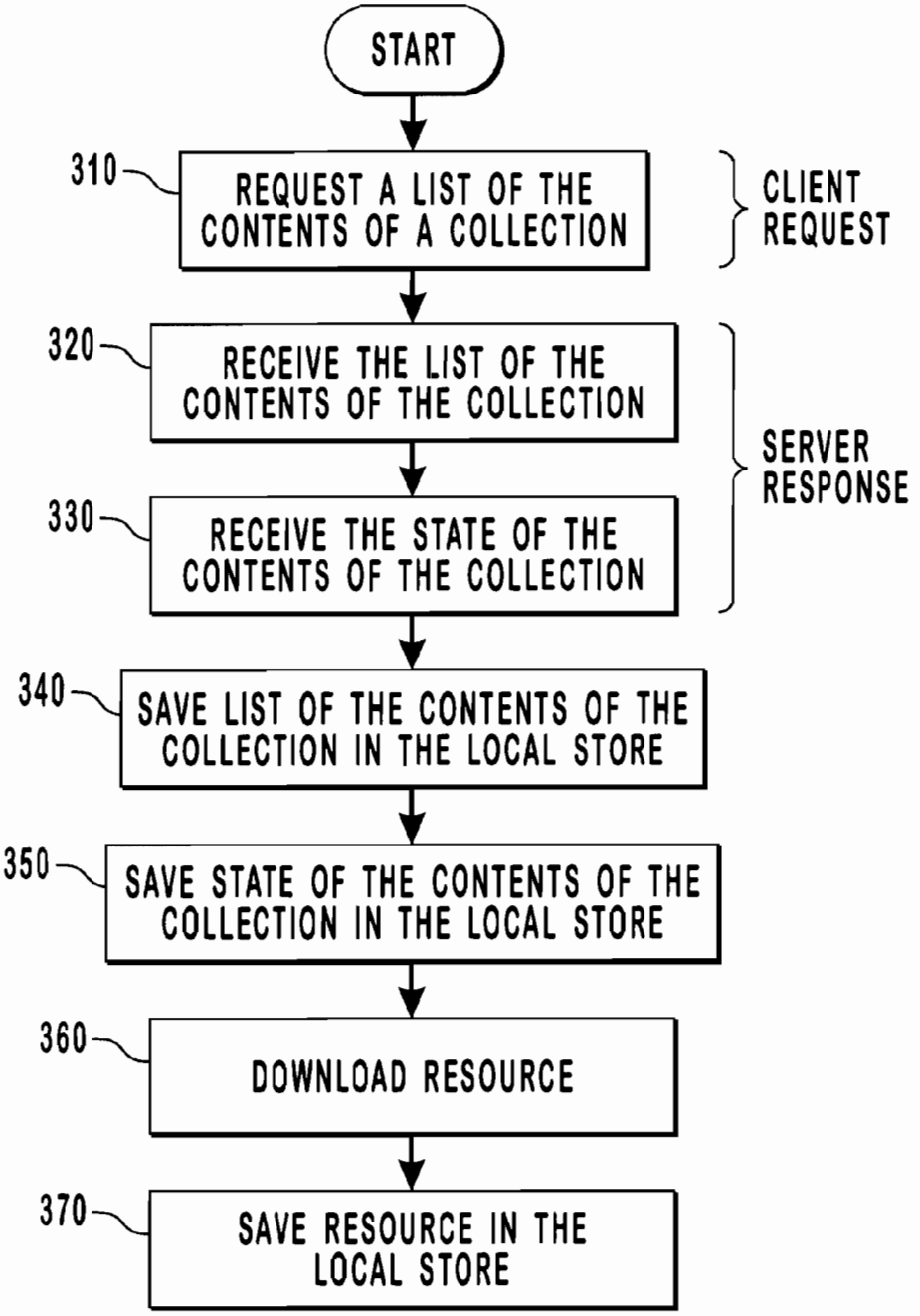

START
310
REQUEST A LIST OF THE CONTENTS OF A COLLECTION
CLIENT REQUEST
320
RECEIVE THE LIST OF THE CONTENTS OF THE COLLECTION
330
RECEIVE THE STATE OF THE CONTENTS OF THE COLLECTION
SERVER RESPONSE
340
SAVE LIST OF THE CONTENTS OF THE COLLECTION IN THE LOCAL STORE
350
SAVE STATE OF THE CONTENTS OF THE COLLECTION IN THE LOCAL STORE
360
DOWNLOAD RESOURCE
370
SAVE RESOURCE IN THE LOCAL STORE

FIG. 3

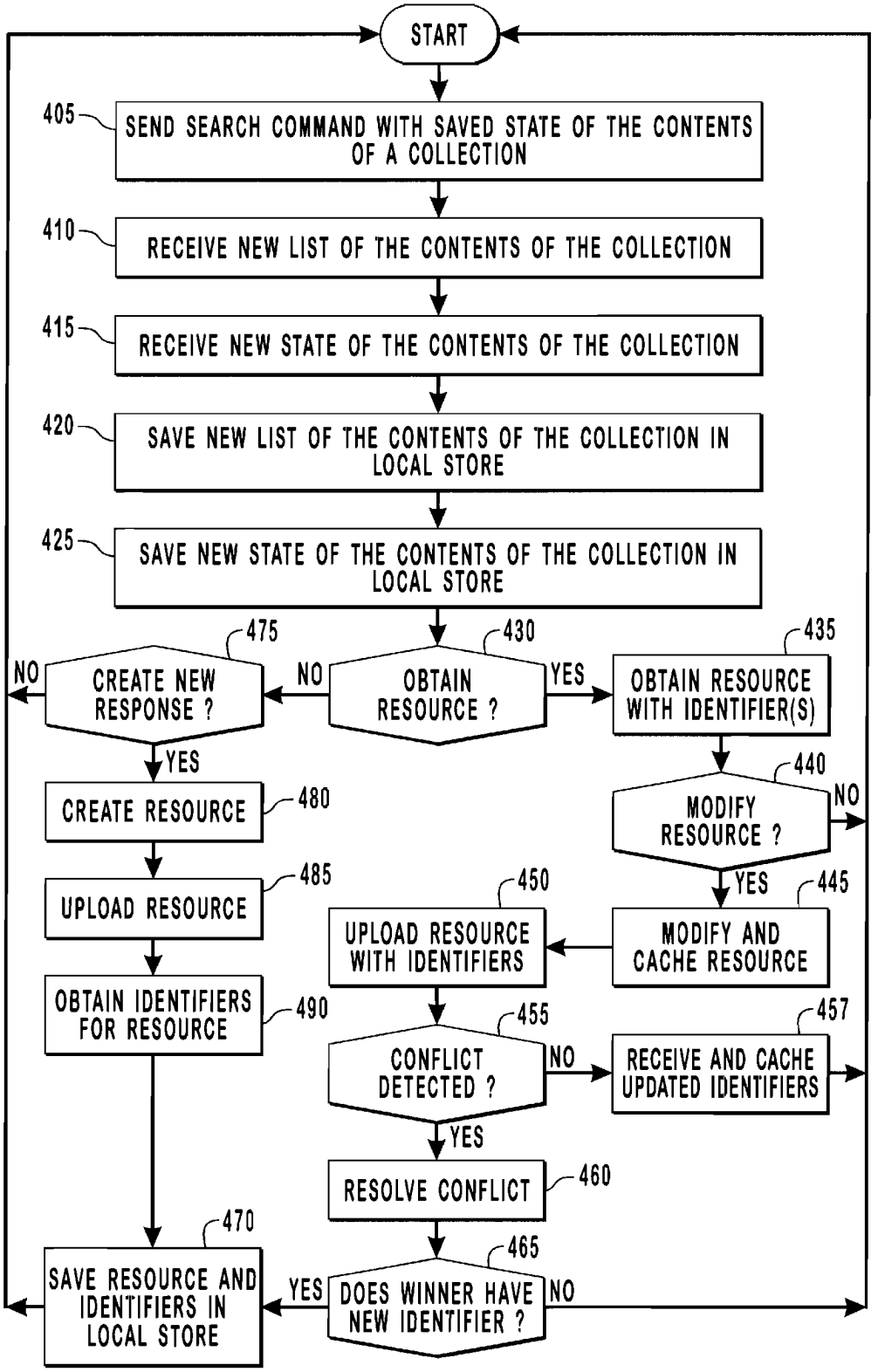

START
405
SEND SEARCH COMMAND WITH SAVED STATE OF THE CONTENTS OF A COLLECTION
410
RECEIVE NEW LIST OF THE CONTENTS OF THE COLLECTION
415
RECEIVE NEW STATE OF THE CONTENTS OF THE COLLECTION
420
SAVE NEW LIST OF THE CONTENTS OF THE COLLECTION IN LOCAL STORE
425
SAVE NEW STATE OF THE CONTENTS OF THE COLLECTION IN LOCAL STORE
475
430
435
NO
CREATE NEW RESPONSE ?
NO
OBTAIN RESOURCE ?
YES
OBTAIN RESOURCE WITH IDENTIFIER(S)
YES
CREATE RESOURCE
480
440
MODIFY RESOURCE ?
NO
485
450
445
UPLOAD RESOURCE
UPLOAD RESOURCE WITH IDENTIFIERS
YES
MODIFY AND CACHE RESOURCE
OBTAIN IDENTIFIERS FOR RESOURCE
490
455
457
CONFLICT DETECTED ?
NO
RECEIVE AND CACHE UPDATED IDENTIFIERS
YES
RESOLVE CONFLICT
460
470
465
SAVE RESOURCE AND IDENTIFIERS IN LOCAL STORE
YES
DOES WINNER HAVE NEW IDENTIFIER ?
NO

FIG. 4

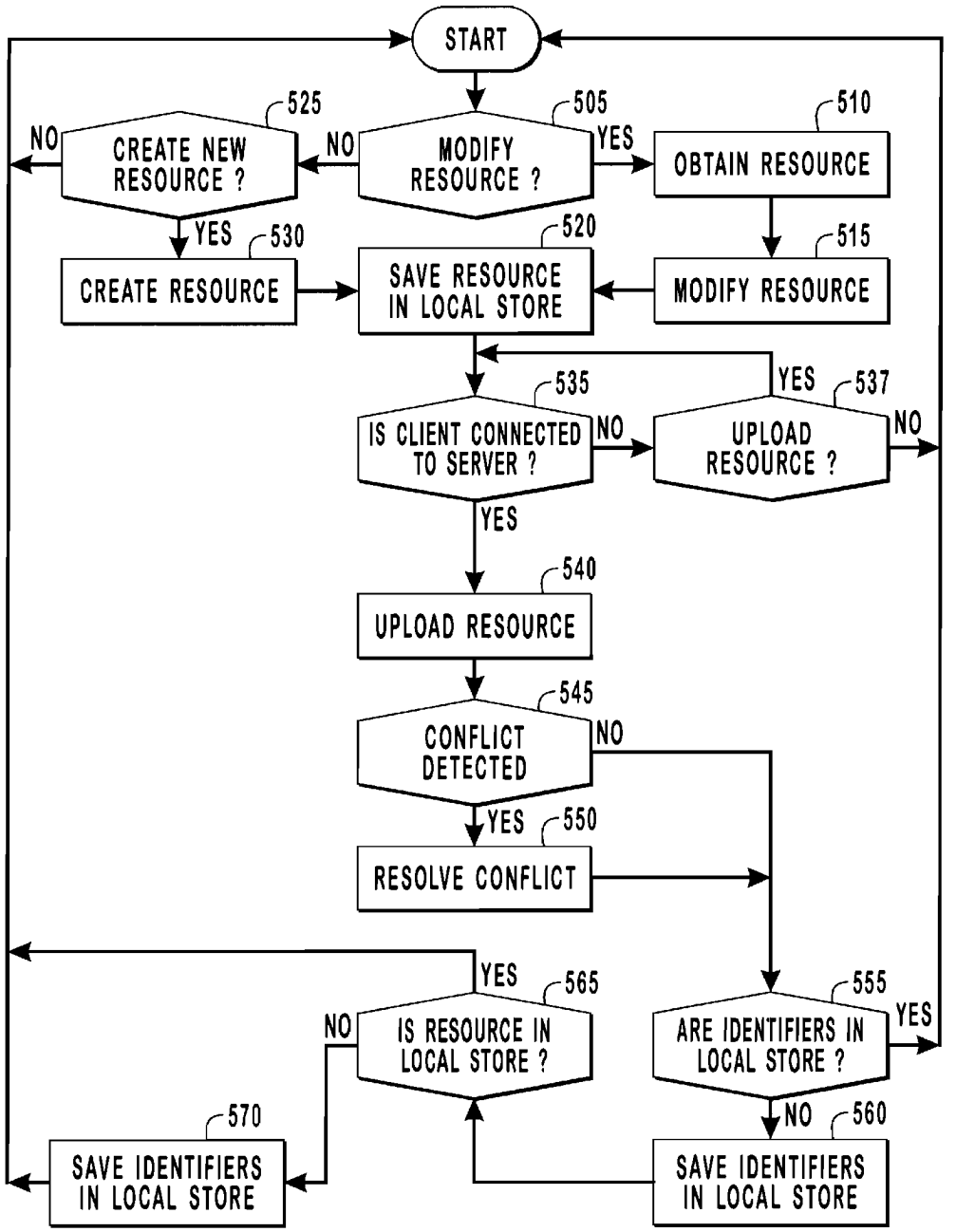

START
505
MODIFY RESOURCE ?
NO
YES
525
CREATE NEW RESOURCE ?
NO
YES
510
OBTAIN RESOURCE
530
CREATE RESOURCE
520
SAVE RESOURCE IN LOCAL STORE
515
MODIFY RESOURCE
535
IS CLIENT CONNECTED TO SERVER ?
NO
YES
537
UPLOAD RESOURCE ?
YES
NO
540
UPLOAD RESOURCE
545
CONFLICT DETECTED
NO
YES
550
RESOLVE CONFLICT
555
ARE IDENTIFIERS IN LOCAL STORE ?
YES
NO
560
SAVE IDENTIFIERS IN LOCAL STORE
565
IS RESOURCE IN LOCAL STORE ?
YES
NO
570
SAVE IDENTIFIERS IN LOCAL STORE

FIG. 5

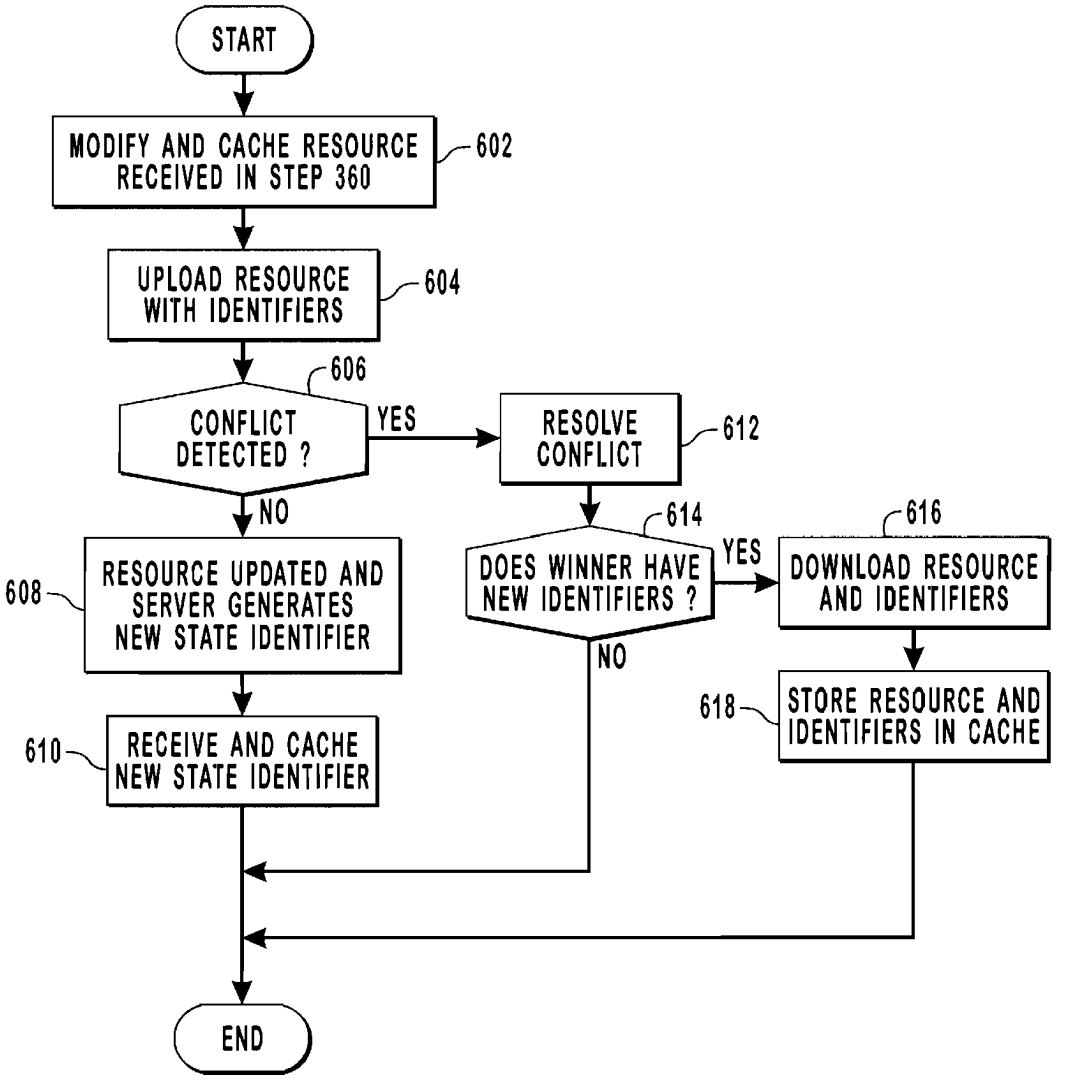

START
MODIFY AND CACHE RESOURCE RECEIVED IN STEP 360
602
UPLOAD RESOURCE WITH IDENTIFIERS
604
606
CONFLICT DETECTED ?
YES
RESOLVE CONFLICT
612
NO
614
DOES WINNER HAVE NEW IDENTIFIERS ?
YES
616
DOWNLOAD RESOURCE AND IDENTIFIERS
608
RESOURCE UPDATED AND SERVER GENERATES NEW STATE IDENTIFIER
NO
618
STORE RESOURCE AND IDENTIFIERS IN CACHE
610
RECEIVE AND CACHE NEW STATE IDENTIFIER
END

FIG. 6

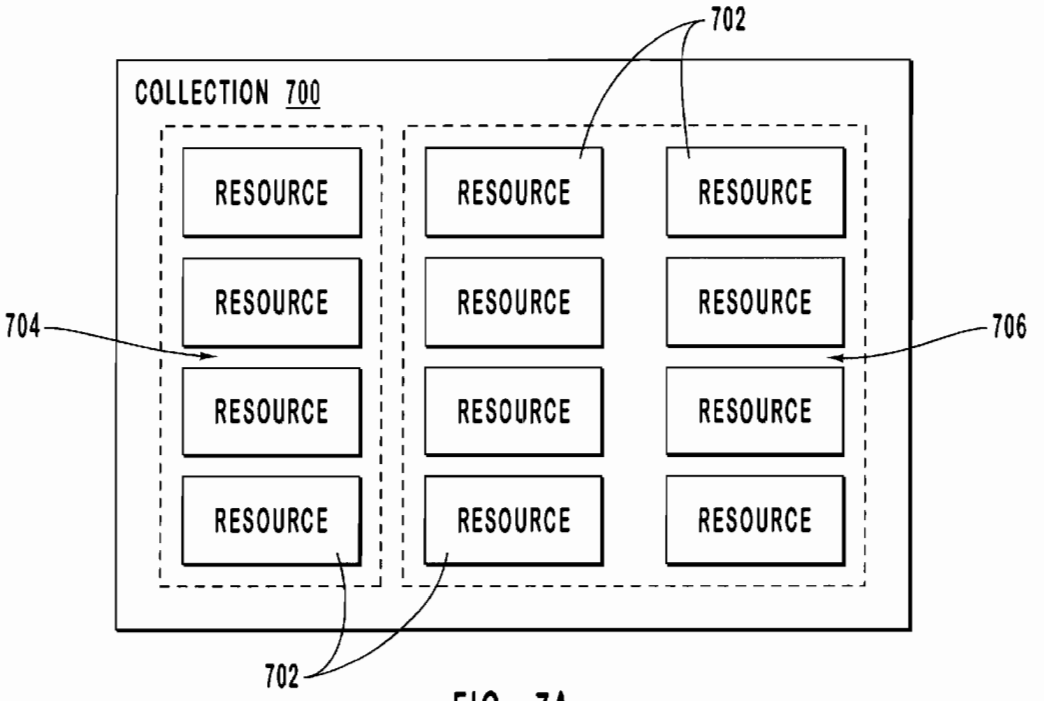

COLLECTION 700
RESOURCE
RESOURCE
RESOURCE
RESOURCE
RESOURCE
RESOURCE
RESOURCE
RESOURCE
RESOURCE
RESOURCE
RESOURCE
RESOURCE
702
702
704
706

FIG. 7A

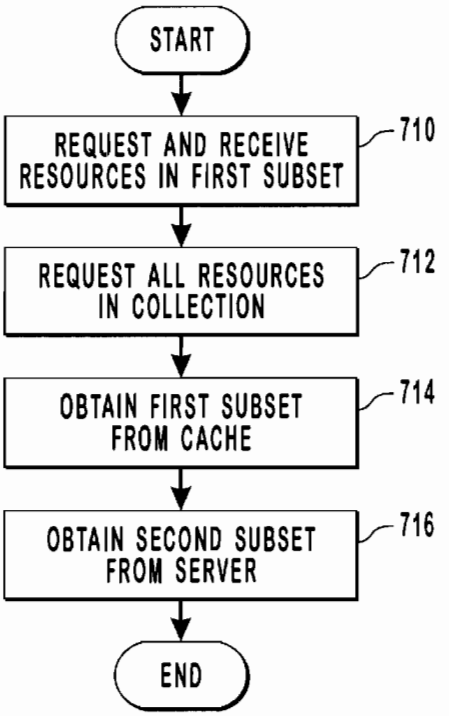

START
REQUEST AND RECEIVE RESOURCES IN FIRST SUBSET
710
REQUEST ALL RESOURCES IN COLLECTION
712
OBTAIN FIRST SUBSET FROM CACHE
714
OBTAIN SECOND SUBSET FROM SERVER
716
END

FIG. 7B

METHOD AND SYSTEM FOR SUPPORTING OFF-LINE MODE OF OPERATION AND SYNCHRONIZATION USING RESOURCE STATE INFORMATION

BACKGROUND OF THE INVENTION

1. The Field of the Invention

The present invention relates to the support of on-line and off-line transmission and synchronization of data. More specifically, the present invention relates to systems and methods that eliminate redundant data transmission and allow multiple copies of data to be synchronized so that incremental changes made to one copy of the data can be identified, transferred, and incorporated into the other copy of the data, regardless of whether the incremental changes are made on-line or off-line.

2. The Prior State of the Art

With the advent of the personal computer as the standard information tool, individuals everywhere at anytime need access to information. Never before has there been so much information readily available or such high expectations for how much the individual will be able to accomplish by utilizing information. It is now more common than ever for multiple users to simultaneously, or in tandem, work on shared data such as, by way of example, word processing documents, electronic mail messages, spreadsheets, electronic forms, graphic images, or a host of other data objects. Thus, accessing and sharing current and accurate information among multiple users has become increasingly vital to businesses and individuals.

Traditionally, a user desiring to access shared information located on one or more servers has had to work "on-line." More specifically, the user logs on to a network and communicates directly with a server. All requests from the user are sent directly to the server. Prior to disconnecting the communication, the user updates the data object on the server and must request a local copy or else the user will not have access to the data object once he or she logs off the network. Once "off-line," the user has no way of knowing whether the local copy of the data object contains current and accurate information because the data object on the server could have been modified by a second user while the first user was off-line. Furthermore, if the data object on the server was modified by a second user while the first user was off-line and if the first user makes changes to his/her local copy of the data object, logs on to the network, and saves the local copy on the server under the original name of the data object, changes made to the data object by the second user will be overwritten and lost.

Another limitation is highlighted when two or more users desire access to a data object simultaneously. Under the traditional method, a first user is allowed to obtain and edit the data object. Any subsequent users desiring access to the data object are limited to a "read only" access of the data object. Therefore, while the first user is using the data object, the subsequent users can access but cannot edit the data object. In order to edit, a subsequent user would be required to save the data object under a different name. Once the subsequent user completes the edits, the changes made to the data object by the first user will be overwritten and lost if the changes made by the subsequent user are saved on the server under the original name. Alternatively, if the subsequent user saves the data object on the server under a new name, multiple copies of the data object will exist on the server. One copy will contain the changes made by the first user and a second copy will contain the changes made by the subsequent user. Where multiple copies exist, it would be difficult to know which copy, if any, contains the most current and accurate information.

Using conventional techniques, clients and servers have engaged in redundant communication of information when a user has been making changes to a shared data object while the client is on-line and subsequently desires to obtain a local copy of the shared data object for use off-line. In particular, the on-line changes made to the shared data object are transmitted from the client to the server during the on-line operation of the client. When the client is about to go off-line, the client issues a request to the server for the most current copy of the shared data object in order to store the copy locally for off-line use. In response, the server transmits the current copy of the data object, which includes the changes that have recently been made by the client. In other words, the foregoing client/server communication involves changes being sent from client to server and subsequently from server to client. Such repetitive transmission of data in a network can introduce potentially significant increases in network traffic, particularly in large organizations with many clients.

It would therefore be desirable to ensure current and accurate information through a model that would synchronize all copies of a data object. It would also be desirable for a synchronization model to be able to identify which copy of a data object is more current and accurate. It would be desirable for none of the changes to be lost or overwritten when multiple copies of a data object are used to update the copy on the server. There is also a need in the art for any such synchronization model to allow multiple users to access and edit the data object simultaneously. Furthermore, it would be desirable if the synchronization mode could eliminate the creation of redundant copies of the data object. It would also be advantageous to eliminate redundant transmissions of data object between clients and servers. Any synchronization model that could exhibit such capabilities would be particularly useful if it could support changes made to local copies of data objects regardless of whether a client was on-line or off-line with the server.

SUMMARY OF THE INVENTION

The foregoing problems in the prior state of the art have been successfully overcome by the present invention, which is directed to a system and method for elimination of redundant data transmission and for incremental change synchronization between multiple copies of data. The systems and methods of the present invention allow multiple copies of data to be synchronized so that incremental changes made to one copy of the data can be identified, transferred, and incorporated into the other copy of the data. The systems and methods of the present invention produce a more efficient use of time and network bandwidth by eliminating the transmission of redundant data. Furthermore, copies of a data object can be reliably and efficiently synchronized regardless of whether the changes made to a copy of the data object are executed while the client is operating on-line or off-line.

Implementation of the present invention may take place in an environment where multiple copies of a data object or resource are present, or where a data object or resource is shared. By way of example, the present invention can be implemented in an environment where a client is connected to a server. The present invention can also be implemented in an environment where multiple clients are connected to

the same server. Another example would include multiple clients connected to multiple servers. In each of the environments, the present invention preserves incremental changes to data object regardless of whether the changes are made while a client is on-line or off-line, ensures that all copies of a data object or resource are synchronized without loss of any incremental changes, and prevents the transmission of redundant data between servers and clients.

As part of the present invention, a client, while connected to a server, identifies to the server the current state of data located at the client and issues a request for the server to evaluate the state of the client's data. The server responds to the request by returning an identification of server data that is not included in the client's data and an identification of the client's data that has been changed on the server. The client is then able to download from or upload to the server new or modified data.

As data is uploaded to or downloaded from the server, the data is saved in a local storage location associated with the client and remains available to the user after the client is no longer connected to the server. In other words, as a client interacts with a server, the client stores a copy of the data object or resource and corresponding identification in cache. Therefore, when the client is "off-line" from the server, the client is still able to work on the data object or resource as if it were "on-line" because a copy is located in the client's cache. Furthermore, the presence of the data object or resource copy in cache prevents the need to download a copy from a server each time that the user desires to obtain the data object or resource. Thus, the repetitive transmission of the same data between a client and server is reduced or eliminated according to the invention.

Later, when the client is again "on-line," all copies of the data object or resource are synchronized. The client identifies to the server the current state of the copy in the client's cache. The server determines if the copy in the client's cache is the most current version of the data object or resource, and all copies of the data object are synchronized to the most current version. If there are conflicts between copies of the data object or resource, the conflicts are detected and resolved and all copies are synchronized to reflect the most current version of the data object or resource.

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

BRIEF DESCRIPTION OF THE DRAWINGS

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

FIG. **1A** illustrates an exemplary system that provides a suitable operating environment for the present invention;

FIG. **1B** is a schematic illustration of an exemplary operating environment that highlights the use of one or more leaf nodes connected to a cloud of servers for implementation of the present invention;

FIG. **2** is a schematic illustration of data transmission between a client and a server that employs a caching mode of operation;

FIG. **3** is a flowchart illustrating an initial synchronization and download during an on-line mode of operation;

FIG. **4** is a flowchart illustrating a method of resource modification, synchronization, download and/or upload performed after the steps of FIG. **3** during an on-line mode of operation; and

FIG. **5** is a flowchart illustrating a method of off-line resource modification and subsequent resource synchronization that is performed after the steps of FIG. **3**.

FIG. **6** is a flowchart depicting a method of modifying the resource downloaded in FIG. **3** and synchronizing the modified resource with a copy stored at the server.

FIG. **7A** is a schematic diagram illustrating a collection and resources contained by the collection.

FIG. **7B** is a flowchart illustrating a method for efficiently downloading a complete copy of the resources in the collection of FIG. **7A** in response by a request by a client.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention extends to systems and methods for the elimination of redundant data transmission in a computer network and for synchronization between multiple copies of data. Multiple copies of data are synchronized so that incremental changes made to one copy of the data can be identified, transferred, and incorporated into all other copies of the data. Eliminating the transmission of redundant data produces a more efficient use of time and network bandwidth. Furthermore, the systems and methods of the present invention support on-line and off-line modes of operation while preserving data transmission efficiency and ensuring reliable synchronization of data.

Embodiments for implementation of the systems and methods of the present invention may comprise a special-purpose or general-purpose computer including computer hardware and/or software components that further include computer-readable media for carrying out or containing computer-executable instructions or data structures. Computer-executable instructions include, for example, instructions and data which cause a special-purpose or general-purpose computer to perform a certain function or group of functions. Computer-readable media can include any available media that can be accessed by a general purpose or special purpose computer. By way of example, and not limitation, such computer-readable media can comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other medium that can be used to carry or store desired program code means in the form of computer-executable instructions or data structures, or that can be used to store information or data, and that can be accessed by a special-purpose or general-purpose computer.

When information is transferred or provided over a means of communication, such as a network (either hardwired, wireless, or a combination of hardwired or wireless), to a computer, the computer properly views the connection as a

computer-readable medium. Thus, any such connection can be properly termed a computer-readable medium.

For purposes of illustration, this description of the invention refers to diagrams and flowcharts depicting the structure or processing of embodiments for implementing the systems and methods of the present invention. Using the diagrams and flowcharts in this manner should not be construed as limiting the scope of the present invention. The description of the invention presented herein makes reference to terminology, methods and concepts from RFC 2518. Therefore, for purposes of the description, RFC 2518 is incorporated herein by reference. Examples of terms used herein and defined in RFC 2518 include "resource" and "collection."

FIGS. **1**A and **1**B provide suitable computing environments in which the invention may be implemented. Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules, being executed by computers in network environments. Generally, program modules include routines, programs, objects, components, data structures, and so forth that perform particular tasks or implement particular abstract data types. Computer-executable instructions, associated data structures, and program modules represent examples of the program code means for executing steps of the methods disclosed herein. The particular sequence of such executable instructions or associated data structures represent examples of corresponding acts for implementing the functions described in such steps.

Those skilled in the art will appreciate that the invention may be practiced in network computing environments with many types of computer system configurations, including personal computers, hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like. The invention is described herein in reference to distributed computing environments where tasks are performed by local and remote processing devices that are linked (either by hardwired links, wireless links, or by a combination of hardwired or wireless links) through a communications or computer network. In the distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to FIG. **1**A, an exemplary system for implementing the invention includes a general purpose computing device in the form of a conventional computer **20**, including a processing unit **21**, a system memory **22**, and a system bus **23** that couples various system components including the system memory **22** to the processing unit **21**. The system bus **23** may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) **24** and random access memory (RAM) **25**. A basic input/output system (BIOS)) **26**, containing the basic routines that help transfer information between elements within the computer **20**, such as during start-up, may be stored in ROM **24**.

The computer **20** may also include a magnetic hard disk drive **27** for reading from and writing to a magnetic hard disk **39**, a magnetic disk drive **28** for reading from or writing to a removable magnetic disk **29**, and an optical disk drive **30** for reading from or writing to removable optical disk **31** such as a CD-ROM or other optical media. The magnetic hard disk drive **27**, magnetic disk drive **28**, and optical disk drive **30** are connected to the system bus **23** by a hard disk drive interface **32**, a magnetic disk drive-interface **33**, and an optical drive interface **34**, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer-executable instructions, data structures, program modules and other data for the computer **20**. Although the exemplary environment described herein employs a magnetic hard disk **39**, a removable magnetic disk **29** and a removable optical disk **31**, other types of computer readable media for storing data can be used, including magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, RAMs, ROMs, and the like.

Program code means comprising one or more program modules may be stored on the hard disk **39**, magnetic disk **29**, optical disk **31**, ROM **24** or RAM **25**, including an operating system **35**, one or more application programs **36**, other program modules **37**, and program data **38**. A user may enter commands and information into the computer **20** through keyboard **40**, pointing device **42**, or other input devices (not shown), such as a microphone, joy stick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit **21** through a serial port interface **46** coupled to system bus **23**. Alternatively, the input devices may be connected by other interfaces, such as a parallel port, a game port or a universal serial bus (USB). A monitor **47** or another display device is also connected to system bus **23** via an interface, such as video adapter **48**. In addition to the monitor, personal computers typically include other peripheral output devices (not shown), such as speakers and printers.

The computer **20** may operate in a networked environment using logical connections to one or more remote computers, such as remote computers **49***a* and **49***b*. Remote computers **49***a* and **49***b* may each be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer **20**, although only memory storage devices **50***a* and **50***b* and their associated application programs **36***a* and **36***b* have been illustrated in FIG. **1**A. The logical connections depicted in FIG. **1**A include a local area network (LAN) **51** and a wide area network (WAN) **52** that are presented here by way of example and not limitation. Such networking environments are commonplace in office-wide or enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, the computer **20** is connected to the local network **51** through a network interface or adapter **53**. When used in a WAN networking environment, the computer **20** may include a modem **54**, a wireless link, or other means for establishing communications over the wide area network **52**, such as the Internet. The modem **54**, which may be internal or external, is connected to the system bus **23** via the serial port interface **46**. In a networked environment, program modules depicted relative to the computer **20**, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing communications over wide area network **52** may be used.

Now referring to FIG. **1**B, an exemplary network environment shown generally at **102** is illustrated. Network **102** includes a cloud of servers **100** that provides network and computing services. The term "cloud of servers" is to be understood as referring to a group of one or more servers, any one of which may be used to provide network and computing services to clients. For purposes of illustration, cloud **100** includes three servers, namely, servers **110**, **120**, and **130**. The number of servers in any particular network in

which the invention is practiced is not critical with respect to the manner in which the invention operates. Depending on the size of network **102**, the capabilities of the servers, and the network services required by the clients, as few as one server or as many servers as needed may be included in cloud **100**. Moreover, the individual servers **110**, **120** and **130** in cloud **100** may be in a single general location in an enterprise, or may be located remotely with respect to one another.

Network **102** also includes one or more clients connected to the cloud of servers **100** by way of a data communication infrastructure **180**, which may be wired or wireless. For illustration purposes, clients **140**, **150**, **160**, and **170**, which may be any special-purpose of general-purpose computers, are depicted in FIG. **1**B. As used herein, the term "leaf nodes" shall refer to clients associated with a network in which the invention is employed. In practice, network **102** can have as few as one client or can have as many clients as is required by the enterprise or organization served by the network. While network **102** might have only one client, the full benefits of the invention are often more pronounced when a plurality of clients are associated with the network, since the invention enables access and synchronization of multiple copies of a particular resource that may be shared among any number of clients. Indeed, one of the advantageous features of the invention is that it can reliably and efficiently manage the synchronization of shared resources, even as the number of clients associated with network **100** grows large.

A client can comprise an internal or external computer readable media for providing local, nonvolatile storage of information, as will be further explained below. Servers **110**, **120** and **130**, are kept in consistent replica of data among themselves so that changes can be entered into any one of the servers. As will be explained below, implementation of the present invention allows for the modifying of a resource by a client regardless of whether that client is logged on or logged off from communication infrastructure **180**, ensures synchronization of all copies of a resource whether at a server or at a client, and optimizes network bandwidth and time by preventing redundant transmission of data across infrastructure **180**.

FIG. **2** illustrates an embodiment of the present invention wherein data is transmitted between a server and a client that employs a caching mode of operation. As mentioned above, a client, such as client **140**, can comprise an internal or external computer readable media, which may be a nonvolatile storage device providing local storage of information. FIG. **2** illustrates the computer readable media as local store **200**.

FIG. **2** illustrates an embodiment of the invention associated with a caching mode of operation in which data is transmitted from client **140** to any one of the servers within the cloud of servers **100**. The data can be sent either by way of direct transmission, or by first passing the data through local store **200**, as depicted by the combination of transmissions **210** and **220**. In a further embodiment, when client **140** is on-line with any one of the servers within the cloud of servers **100**, all data transmission is sent directly to the server. When client **140** is off-line with servers **100**, all data operations, such as read requests, write requests, delete requests and other communication generated for accessing or manipulating a resource is communicated between client **140** and local stored **200**.

As used herein, the term "on-line" refers to a state in which a client communicates with a server using the communication infrastructure. For example, a client that is sending network requests to a server or receiving network or computing services from the server is on-line. In contrast, the term "off-line" refers to a client is operating in a state in which it is not immediately capable of actively communicating with a server. For instance, a client that has physically disconnected from a wired communication infrastructure or is not communicating over a wireless communication infrastructure is off-line. Moreover, a client that may be physically connected to a communication infrastructure, but is operating only locally without communicating over the infrastructure can also be considered to be off-line.

In contrast to the communication from client to server illustrated in FIG. **2**, all data transmission originating from a server is first passed through local store **200** before reaching client **140** in this embodiment. In this manner, local store **200** can cache all data transmitted from the cloud of servers **100** to client **140**. As such, when client **140** is off-line, it can obtain the data cached in local store **200** in a similar way as if client **140** were on-line with any one of the servers within the cloud of servers **100**, such as server **110**, and obtained the data directly from server **110**. Therefore, when client **140** requests data that has been cached in local store **200**, it can obtain the data regardless as to whether client **140** is on-line or off-line. Indeed, from the standpoint of client **140**, retrieval of a resource from local store **200** during an off-line mode of operation can appear as if the client remained on-line and were accessing the resource from servers **100**.

Another embodiment of the invention is associated with a caching mode of operation, In this embodiment the data can only be sent to one of the servers, such as server **110**, by first sending the data to local store **200**, as depicted by the combination of transmissions **210** and **220**. Therefore, local store **200** is involved in all communication with servers **100** while client **140** is on-line and all read requests, write requests, or any other communication generated for accessing or manipulating a resource while the client is off-line.

Similarly, all data transmission originating from a server is first passed through local store **200** before reaching client **140**. In this manner local store **200** can cache all data transmitted from the cloud of servers **100** to client **140**. As such, when client **140** is off-line, it can obtain the data cached in local store **200** in a similar way as if client **140** were on-line with any one of the servers within the cloud of servers **100**. Therefore, when client **140** requests data that has been cached in local store **200**, it can obtain the data regardless as to whether client **140** is on-line or off-line.

As introduced above, the systems and methods of the present invention provide for the synchronization between multiple copies of data. Multiple copies of data are synchronized so that incremental changes made to one copy of the data can be identified, transferred, and incorporated into all other copies of the data.

FIG. **3** illustrates one embodiment of a method for performing initial synchronization between a client and server and download during an on-line mode of operation. The method of FIG. **3** can be used to coordinate the use and modification of any type of resource. However, in order establish a context in which the following description can be easily understood and not by limitation, the resources could be a set of e-mail messages stored at the server, while the collection could be an inbox that stores the e-mail messages or the resources could be a set of word processing documents, while the collection could be a file system folder that stores the word processing documents. In the following

description associated with FIGS. **3–5**, specific reference will be made to word processing documents to illustrate this embodiment of the invention. Again, however, the invention can be practiced with any desired type of resource.

In step **310** a client requests a list of the contents of a collection located at a server. The server then responds by transmitting a list of the resources contained in the collection and information that specifies the state of the resources contained in the collection, as respectively illustrated by steps **320** and **330**. In step **320** the client receives the list of the contents of the collection. In one embodiment, the list includes identifiers representing each of the resources of the collection. Although not necessary, the identifiers can be compressed prior to transmission to optimize network bandwidth and time. The identifiers can include, for example, a Uniform Resource Identifier (URI) or other information that uniquely identifies the resources.

In step **330** the client receives the information specifying the state of the contents of the collection. The "state" of the contents of a collection stored at a server refers to the identity of the current version of a resource stored at a server. The state of any particular copy of a resource refers to the version of the resource when it was stored at the server. As a particular resource stored at one or more servers undergoes a series of successive updates, the resource is considered to have passed through a corresponding series of states, each of which represents a single update version of the resource. In one embodiment, information specifying the state of the contents of the resources includes an identifiers, which can be termed resource state identifiers. Although not necessary, these identifiers can also be compressed to optimize network bandwidth and time.

In summary, steps **310**, **320**, and **330** result in the client being given at least two pieces or sets of information associated with the collection specified in the request of step **310**. First, the client is given information representing the identity of the various resources that are contained in a collection stored at the server in step **320**. Second, the client is given information that essentially represents or is associated with the current update version of the various resources that are contained in the collection in step **330**. These two pieces or sets of information can be subsequently used by the methods and systems of the invention to manage synchronization of copies of the resources contained in the collection and to eliminate repetitive transmission of resources or portions of resources between the client and the server as will be further described herein below.

In order to obtain a more detailed description of the processes and mechanisms whereby steps **310**, **320**, and **330** can be performed, reference is made to U.S. patent applications Ser. No. 09/412,739, entitled "Method, Computer Readable Medium, and System For Monitoring The State of A Collection of Resources", filed on Oct. 4, 1999, which assigned to the same assignee as the present application, and which is incorporated herein by reference; and Ser. No. 09/412,071, entitled "Method, Data Structure, and Computer Program Product For Identifying A Network Resource", filed on Oct. 4, 1999, which assigned to the same assignee as the present application, and which is incorporated herein by reference.

In steps **340**, **350** and **370** data is cached in a local storage associated with the client. More specifically, in step **340**, the list of the contents of the collection received in step **320** is cached in a local storage. In step **350** the state of the contents of the collection received in step **330** is cached in local storage. The client can then obtain one or more resources from the server. In step **360** the client requests and the server downloads a resource and corresponding identifiers in the collection. If, for example, the resources in the collection are word processing documents, the client in **360** can request a selected document. The corresponding identifiers referred to in step **360** can include information that represents the current update version of the downloaded resource. In step **370** the downloaded resource and corresponding identifiers are cached in local storage. For purposes of this description and the accompanying claims, the term "download" refers to data transmission from server to client, whereas "upload" refers to data transmission from client to server.

It should be noted that steps **360** and **370** are optional, depending on whether the user of the client wants to obtain a particular resource at this time. If no resource is downloaded at this time, the client still has obtained information in steps **320** and **330** that identifies the resources contained in the collection and represents the current update version of the resources as they are stored at the server.

FIGS. **4–6** represent various methods and operations that can be performed in the network after the initial synchronization of FIG. **3**. FIG. **4** represents a method of performing on-line modification, downloading, uploading, and synchronization of resources. FIG. **5** depicts a method whereby a client modifies a resource off-line, which is later synchronized on-line. Finally, FIG. **6** represents a method of modifying and synchronizing the resource downloaded in step **360** of FIG. **3**.

FIG. **4** provides a flowchart illustrating a subsequent resource modification, synchronization, download and/or upload during an on-line mode of operation. The mode of operation in FIG. **4** is "on-line" in the sense that any modification of resources is conducted while the client is on-line. The method illustrated in FIG. **4** is conducted after an initial synchronization has been conducted as shown, for example, in FIG. **3** and assumes that the client operates in the caching mode.

In step **405** of FIG. **4**, the client again requests a list of the contents of a collection located at the server. The server then responds by transmitting the list and state of the contents of the collection, as respectively illustrated by steps **410** and **415**. In step **410** the client receives a new list of the contents of the collection. Depending on the activity associated with the collection that has been conducted at the server since the time of the previous request (i.e., step **310** of FIG. **3**), the list obtained in step **410** can be identical or different from the list obtained in step **320** of FIG. **3**. For instance, if a word processing document has been deleted or created at the server (perhaps by a different client) since the time of step **310**, the list obtained in step **410** will include information identifying the new word processing document. In contrast, if no changes to the collection have been made, the list received in step **410** will be identical to the list obtained in **320**.

In step **415** the client receives a new state of the contents of the collection. In a similar manner, the new state can be identical to the state previously received in step **330** of FIG. **3** or can be an updated version. For example, if an existing word processing document has been modified and stored at the server (perhaps by a different client) since the time of the previous request (i.e., step **310**), the state of the modified word processing document will be different, reflecting the updated version of the document. Similarly, if a word processing document has been created or deleted at the server since the time of step **310**, the state of the contents of the collection will also be different. In contrast, if no updated

versions have been stored and no resources have been created or deleted, the state received in step **415** will be identical to the list received in step **330**.

Assuming that the new list and the new state are differ from the previous list and state, steps **420** and **425** cache the new list and new state, respectively, in local storage. Decision block **430** then inquires as to whether the client desires to obtain a resource from the server. If the client does, the method advances to step **435**. Otherwise, the method proceeds to decision block **475**.

In step **435** the client requests a resource and the identifiers corresponding to the resource. In this step and others that follow, unless otherwise indicated, the "identifiers" refer to at least information that uniquely identifies the resource and information that specifies the update version of the resource. If the resource has never been cached in local storage, the resource and corresponding identifiers are downloaded from the server in step **435**. In this manner, repetitive downloading of the same resource to a particular client is eliminated.

Alternatively, if the resource has been previously cached in the local store associated with the requesting client, step **435** involves a comparison that takes place between the state of the resource in the local store with the state of the resource on the server. This comparison is conducted in order to determine whether the copy of the resource stored in the local store is the most recent version or whether a more recent version exists at the server. In other words, the comparison addresses the possibility that another client has modified and updated the requested resource since the last time that the requesting client has obtained the copy of the resource. It should be noted that the comparison requires transmission of an identifier representing the state of the resource, without requiring transmission of the entire resource between client and server. In this manner, the comparison reduces the network traffic that might otherwise be required and avoids transmitting the same version of the resource more than once.

A more detailed explanation of how the comparison can be performed is included in U.S. patent application Ser. No. 09/412,739 entitled "Method, Computer Readable Medium, and System For Monitoring The State of A Collection of Resources", which was previously incorporated herein by reference; and in U.S. patent application Ser. No. 09/412,738, entitled "Systems and Methods for Detecting and Resolving Resource Conflicts" which was filed on Oct. 4, 1999, which was assigned to the same assignee as the present application, and which is incorporated herein by reference.

If the resource at the local store is found to have an identical or more recent state compared to the state of the resource located at the server, then the client obtains the resource in step **435** by accessing the locally stored resource rather than the copy stored at the server, thereby optimizing network bandwidth and time. Otherwise, the resource and corresponding identifiers are downloaded in step **435** from the server and cached in the local store, thereby replacing the stale resource and identifiers that have been stored locally.

Decision block **440** inquires as to whether the client is to modify the resource. For instance, the client is to modify the resource if the user wants to edit the contents of the resource or delete the resource. If the client does not desire to modify the resource, processing returns back to step **405**. Otherwise, the client modifies the resource in step **445** and it is cached in local storage.

After the modifications have been performed, the client in step **450** can update the copy of the resource at the server by uploading the resource with its corresponding identifiers. Upon receiving the uploaded resource and corresponding identifiers, the server in decision block **455** compares the state of the uploaded resource with the copy of the resource located at the server to determine whether there is a conflict. In other words, the comparison determines whether the copy of the resource stored at the server has been modified during the period of time that has elapsed since the uploading client last accessed the resource from the cloud of servers. To illustrate further, a conflict may arise when two clients access the same copy of a word processing document stored at a server and both attempt to make and save modifications to the document.

If there is no conflict, the server updates the version of the resource and the identifiers stored at the server and, in step **457**, the updated identifiers are received by the client and cached in local storage. Processing then returns to step **405**. If a conflict is detected then the conflict is resolved in step **460**. A server or a client can perform conflict resolution. Alternatively, a user can be prompted to resolve the conflict. A more detailed description of conflict resolution that can be used with the invention is included in U.S. patent application Ser. No. 09/412,738, entitled "Systems and Methods for Detecting and Resolving Resource Conflicts." Once the conflict has been resolved, decision block **465** inquires as to whether the winning, or more current, resource has corresponding identifiers that are different from the corresponding identifiers located in cache. If the identifiers are identical then processing returns to step **405**. Otherwise, the winning resource and corresponding identifier are cached in local storage in step **470**.

Returning to decision block **430**, if it is determined that the client does not desire to obtain a resource then decision block **475** determines whether the client desires to create a new resource. If the client does not desire to create a new resource then processing returns to step **405**. Otherwise, the client creates a new resource in step **480** and the new resource is uploaded to the client in step **485**. In step **490** the server assigns identifiers to the new resource and downloads the identifiers to the client. It should be noted that, in this embodiment, the identifiers assigned by the server to the new resource is downloaded to the client in step **490**, but the resource is not downloaded, since it already exists at the client. The new resource and corresponding identifiers are cached in local storage in step **470** and processing returns to step **405**.

FIG. **5** provides a flowchart illustrating resource modification, synchronization, download and/or upload, some of which can occur during an off-line mode of operation. The mode of operation in FIG. **5** is "off-line" in the sense that modification of resources can be conducted while the client is off-line. For instance, a network user can download a resource, disconnect the client from the network, make modifications to the resource while off-line, and later synchronize while on-line. The method illustrated in FIG. **5** is conducted after an initial synchronization is performed on-line as shown, for example, in FIG. **3**. FIG. **5** further assumes that the client operates in the caching mode.

For purposes of illustration, it is presumed that the client begins off-line at decision block **505**, which **505** inquires as to whether the client desires to modify a resource. If the client does not desire to modify the resource, then processing proceeds to decision block **525**. Otherwise, the client requests and obtains a resource in step **510**. Because a presumption is being made that the client is off-line, the client obtains the copy of the requested resource stored in the local store associated with the client in step **510**, if such a

copy is stored locally. If the resource is not yet cached in the local store, an error is displayed and processing returns to decision block **505**. The client modifies the resource in step **515** and it is cached in step **520**.

Returning briefly to step **510**, if the client had instead been on-line and the requested resource had been stored locally, a comparison would have been made regarding the state of the local copy of the resource and the state of the copy of the resource stored at the server. If the state of the resource stored at the server demonstrated that the server's copy of the resource was more current than the cached copy of the resource, the server's copy and corresponding identifiers would have been downloaded to the client and cached.

Returning now to the off-line presumption and to decision block **505**, if the client does not desire to modify the resource, decision block **525** inquires as to whether the client desires to create a new resource. If a new resource is not to be created then processing returns to decision block **505**. Otherwise, the new resource is created in step **530** and cached in step **520**.

After the resource is cached in step **520**, decision block **535** inquires as to whether the client is on-line. If it is not on-line, decision block **537** asks whether the client desires to upload the resource to the server. Processing returns to decision block **505** if the resource is not to be uploaded. Otherwise, processing remains in a loop between decision blocks **535** and **537** until the client is on-line with the server.

Once the client is on-line, the client uploads the resource and the corresponding identifiers to the server in step **540**. Upon receiving the uploaded resource and corresponding identifiers the server in decision block **545** compares the state of the uploaded resource with the copy of the resource located at the server, if any. If there is no conflict then the server updates the identifiers and decision block **575** asks whether the corresponding identifiers have been cached.

If a conflict is detected in decision block **545** then the conflict is resolved in step **550**. A server or a client can perform conflict resolution. Alternatively, a user can be prompted to resolve the conflict. A more detailed description of conflict resolution that can be used with the invention is included in U.S. patent application Ser. No. 09/412,738, entitled "Systems and Methods for Detecting and Resolving Resource Conflicts."

Decision block **555** inquires as to whether the identifiers corresponding to the winning, or more current, resource are located in cache. If the identifiers are in cache then processing returns to decision block **505**. Otherwise, the identifiers are cached in step **560**. Decision block **565** inquires as to whether the winning, or more recent, resource is in cache. If it is then processing returns to decision block **505**. Otherwise the resource is cached and processing returns to decision block **505**.

FIG. **6** illustrates selected steps of a method for modifying and synchronizing the resource that was received by the client in step **360** of FIG. **3**. The method illustrated in FIG. **6** is performed after an initial synchronization has been conducted as shown, for example, in FIG. **3** and assumes that the client operates in the caching mode. Step **602** can be executed while the client is on-line or off-line.

In step **602**, the client, either in the on-line of off-line mode, modifies the resource received in step **3** of FIG. **3** and stores the changes to the resource in cache in the local store. Later, when the client is on-line and is to initiate synchronization of the resource, it uploads the resource and the corresponding identifiers, including the information (obtained in step **360**) that specifies the update version of the copy of the resource stored at the server.

In decision block **606**, the server then performs conflict detection as disclosed previously in reference to step **455** of FIG. **4**. If no conflict is detected or, in other words, if the copy of the resource to be overwritten at the server has not changed since the time it was downloaded to the client in step **360** of FIG. **3**, the server updates the resource in step **608** and generates a new state identifier, which specifies that a new updated version of the resource has been stored at the server. In step **610**, the client receives and caches the new state identifier. It is noted that the server does not need to transmit the updated copy of the resource in this embodiment, since the client has already cached this version of the resource in step **602**.

Returning to decision block **606**, if a conflict is detected, the method advances to step **612**, in which the conflict is resolved. A server or a client can perform conflict resolution. Alternatively, a user can be prompted to resolve the conflict. A more detailed description of conflict resolution that can be used with the invention is included in U.S. patent application Ser. No. 09/412,738, entitled "Systems and Methods for Detecting and Resolving Resource Conflicts." Once the conflict has been resolved, decision block **614** inquires as to whether the winning, or more current, resource has new identifiers that are different from the corresponding identifiers located in cache. If the identifiers corresponding to the winning resource are not new, the method illustrated in FIG. **6** ends. Otherwise, the winning resource and corresponding identifiers are downloaded to the client in step **616** and cached in local storage in step **618**.

FIGS. **7A** and **7B** illustrate one example whereby the invention efficiently uses time and network bandwidth. FIG. **7A** depicts a collection **700** that has a plurality of resources **702** contained therein that can be stored at the one or more servers included in the cloud of servers depicted in FIG. **1B**. For purposes that will become clear below in reference to FIG. **7B**, resources **702** are arbitrarily divided into two mutually exclusive subsets, namely, subset **704** and subset **706**. As noted previously, the invention can be practiced with any desired collections and resources. However, in order to establish an clearly understandable context in which FIG. **7B** can be described, it will be assumed that resources **702** are e-mail messages, while collection **700** is an inbox that contains the e-mail messages.

FIG. **7B** is a flowchart illustrating an example in which a client has been accessing a first set of resources in a collection while on-line and, in preparation for going off-line, requests a copy of all the resources in a collection. Rather than download all of the resources in the collection, the server downloads only those that have not yet been accessed by the client, since the previously accessed resources are cached in the local store associated with the client. The server uses the techniques disclosed above in reference to FIGS. **3–6** to determine which resources have been previously accessed by the client and which are to be downloaded, and these techniques are further described below in reference to the specific steps of FIG. **7B**.

In step **710**, the client requests only selected resources in collection **700**. One specific example of selecting only some resources in a collection might happen when the user of the client is interested in accessing only specific e-mail messages stored in an inbox. For instance, in step **710**, the client might request the first subset of resources **704** illustrated in FIG. **7A**. While not explicitly illustrated in FIG. **7A**, the server transmits the identifiers disclosed above in reference to steps **310**, **320**, and **330** of FIG. **3** during step **710** for all of the resources **702** of collection **700**. Moreover, the downloaded first set of **323** resources **704** are cached in the

local store associated with the client, meaning that these same resources will not need to be sent a second time to the client.

In step **712**, the client requests all of resources **702** in collection **700**. Such a request might occur, for example, when the user of the client is preparing to go off-line and wishes to have access to all the e-mails in the inbox. To execute the request, the client and server perform the operations disclosed above in reference to step **435**. In particular, the client obtains the first subset of resources **704** from the cache and the second subset of resources from the server. Moreover, if any of the resources of the first subset have changed at the server since the time at which step **710** was executed, the server downloads those changed resources to the client as further described above in reference to step **435**. In any event, no single version of a resource is sent more than once from the server to the client. In view of this example, it can be clearly seen that the systems and methods of the invention can significantly reduce the time and network traffic that might otherwise be associated with uploading, downloading, and synchronization of resources on a network.

Therefore, as described herein, the systems and methods of the invention eliminate transmission of redundant data by assigning and comparing identifiers for resources to ensure that a resource is transmitted only once to a specific location, thereby producing a more efficient use of time and network bandwidth. The invention also synchronizes multiple copies of a resource by assigning identifiers, comparing identifiers, determining which is the most current resource, and updating the stale resources. Furthermore, no incremental change is lost because resources that are the most current are maintained and because conflicts between resources are detected and resolved. Furthermore, the systems and methods of the present invention support modification of resources while clients are in on-line and off-line modes of operation. Moreover, when a client requests data that has been cached in a local store, it can obtain the data regardless as to whether the client is "on-line" or "off-line."

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 and desired to be secured by United States Letters Patent is:

1. In a networked system including a server and a client, wherein the server stores a resource, a method of enabling a client to access the resource, comprising the steps of:

transmitting, from the server to the client, resource state information that represents the state of the resource stored at the server at a selected moment;

transmitting, from the server to the client, a copy of the resource as it exists at the server at the selected moment;

storing the copy of the resource in a local store associated with the client;

modifying the copy of the resource stored in the local store;

transmitting the modified copy and the resource state information from the client to the server;

based on the resource state information transmitted to the server, determining whether the resource stored at the server has changed since the selected moment; and

if it is determined that the resource stored at the server has not changed, replacing the resource stored at the server with the modified copy.

2. A method as recited in claim **1**, wherein the local store is a nonvolatile storage device that is used as a cache.

3. A method as recited in claim **1**, wherein the step of modifying is conducted by the client while the client is off-line with respect to the server.

4. A method as recited in claim **1**, further comprising, after the step of replacing, the step of transmitting new resource state information to the client.

5. A method as recited in claim **1**, wherein the step of transmitting the modified copy comprises the step of storing the modified copy at the local store without transmitting the modified copy from the server to the client.

6. A method as recited in claim **1**, wherein the step of determining whether the resource stored at the server has changed since the selected moment comprises the step of comparing, at the server, the resource state information transmitted to the server with a current version of the resource state information, wherein the current version of the resource state information represents the state of a version of the resource stored at the client at the time of the step of comparing.

7. A method as recited in claim **6**, wherein the step of comparing results in a determination that the resource state information transmitted to the server and the current version of the resource state information are the same, thereby determining that the resource stored at the server has not changed.

8. A method as recited in claim **6**, wherein the step of comparing results in a determination that the resource state information transmitted to the server and the current version of the resource state information are different, thereby determining that the resource stored at the server has changed, the method further comprising the step of performing a conflict resolution operation in response to the resource stored at the server having changed.

9. A method as recited in claim **1**, further comprising the step of storing, in the local store, the resource state information.

10. A method as recited in claim **1**, wherein step of transmitting the copy of the resource from the server to the client is performed only once.

11. In a client associated with a networked system that includes a server that stores a resource, a method for accessing the resource and then interacting off-line with the resource in a manner such that it appears, from the standpoint of the client, that the client is on-line, comprising the steps of:

while the client is on-line with the server, receiving from the server a copy of the resource and resource state information representing the state of the resource stored at the server at a selected moment;

storing the copy of the resource in a local store associated with the client;

placing the client in an off-line condition with respect to the server; and

performing a data operation on the copy of the resource while the client is off-line by accessing the copy of the resource in the local store, the data operation resulting in a modified copy of the resource; and

synchronizing the resource stored at the server with the modified copy of the resource while the client is subsequently on-line after the step of performing the data operation, the synchronization being performed at

least in part by transmitting to the server the copy or the resource stored at the client, and the resource state information.

12. A method as recited in claim **11**, further comprising the steps of:

determining whether the resource stored at the server has changed since the selected moment based on the resource state information transmitted to the server; and

if it is determined that the resource stored at the server has not changed, replacing the resource stored at the server with the modified copy.

13. A method as recited in claim **11**, wherein the data operation includes a read operation.

14. A method as recited in claim **11**, wherein the data operation includes a write operation.

15. A method as recited in claim **11**, wherein the data operation includes a delete operation.

16. A computer program product for implementing, in a server included in a network that also includes at least a first client and a second client, a method for synchronizing multiple copies of a resource that are stored at various locations in the network, the computer program product comprising:

a computer-readable medium carrying computer-executable instructions for implementing the method, the computer-executable instructions comprising:

program code means for generating resource state information that represents the current state of a resource stored at the server;

program code means for transmitting, from the server to the first client, a copy of the resource state information as it exists at the server at the selected moment and a copy of the resource as it exists at the selected moment;

program code means for receiving a request to update the resource stored at the server with a modified copy of the resource generated by the first client, the request including the copy of the resource state information;

program code means for determining whether a conflict exists in response to the request by comparing the copy of the resource state information received from the first client with the resource state that represents the current state of the resource stored at the server; and

program code means for replacing the resource stored at the server with the modified copy if it is determined that no conflict exists.

17. A computer-readable medium as recited in claim **16**, wherein the computer-executable instructions further comprise program code means for performing conflict resolution if it is determined that a conflict exists.

18. A computer-readable medium as recited in claim **16**, wherein the request to update the resource is encoded in an XML language.

19. A computer program product for implementing, in a server included in a network that also includes at least a first client and a second client, a method for synchronizing multiple copies of a resource that are stored at various locations in the network, the computer program product comprising:

a computer-readable medium carrying computer-executable instructions for implementing the method, the computer-executable instructions comprising:

program code means for generating resource state information that represents the current state of a resource stored at the server;

program code means for transmitting, from the server to the first client, a copy of the resource state information as it exists at the server at the selected moment and a copy of the resource as it exists at the selected moment;

program code means for receiving a request to update the resource stored at the server with a modified copy of the resource generated by the first client, the request including the copy of the resource state information;

program code means for determining whether a conflict exists in response to the request by comparing the copy of the resource state information received from the first client with the resource state that represents the current state of the resource stored at the server; and

program code means for performing conflict resolution if it is determined that a conflict exists.

20. A computer-readable medium as recited in claim **19**, wherein the request to update the resource is encoded in an XML language.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,578,054 B1
DATED : June 10, 2003
INVENTOR(S) : Hopmann et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 2,
Line 37, after "missions of" please insert -- the --

Column 3,
Line 4, after "changes to" please insert -- the --
Line 62, after "that these" please change "drawing" to -- drawings --

Column 5,
Line 55, after "(BIOS)" please delete the second instance of ")"

Column 8,
Line 31, after "mode of operation" please change "," to -- . --
Line 60, after "in order" please insert -- to --

Column 9,
Line 29, after "includes an" please remove "an"

Column 13,
Line 60, after "in the on-line" please change "of" to -- or --

Column 14,
Line 40, after "to establish" please change "an" to -- a --
Line 67, after "first set of" please remove "323"

Column 16,
Line 42, after "wherein" please insert -- the --

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

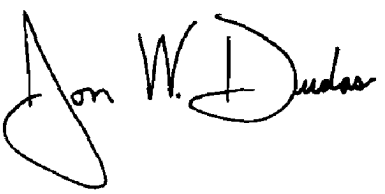
PATENT NO. : 6,578,054 B1
DATED : June 10, 2003
INVENTOR(S) : Hopmann et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 17,
Line 1, after "server the copy" please change "or" to -- of --

Signed and Sealed this

Thirteenth Day of July, 2004



JON W. DUDAS
Acting Director of the United States Patent and Trademark Office

US006370566B2

(12) **United States Patent**
Discolo et al.

(10) **Patent No.: US 6,370,566 B2**
(45) **Date of Patent: *Apr. 9, 2002**

(54) **GENERATING MEETING REQUESTS AND GROUP SCHEDULING FROM A MOBILE DEVICE**

(75) Inventors: **Anthony Discolo**, Redmond; **Scott Skorupa**, Newcastle; **Salim Alam**, Redmond; **Garrett R. Vargas**, Kirkland; **Dave Whitney**, Bellevue; **Bryce Ulrich**, Kirkland; **John I. Ferrell**, Bellevue, all of WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/058,679**

(22) Filed: **Apr. 10, 1998**

(51) **Int. Cl.**[7] **G06F 15/16**
(52) **U.S. Cl.** **709/206**; 709/201; 709/203
(58) **Field of Search** 709/106, 206, 709/201, 227, 216, 10, 203

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,392,390 A		2/1995	Crozier	395/161
5,630,081 A		5/1997	Rybicki et al.	395/348
5,664,228 A		9/1997	Mital	395/882
5,666,530 A	*	9/1997	Clark et al.	707/201
5,684,990 A		11/1997	Boothby	395/619
5,701,423 A		12/1997	Crozier	395/335
5,729,687 A	*	3/1998	Rothrock et al.	709/205
5,758,354 A	*	5/1998	Huang et al.	709/106
5,805,830 A	*	9/1998	Reese et al.	709/205
5,832,489 A	*	11/1998	Kucala	707/10
5,856,978 A	*	1/1999	Anthias et al.	370/429
5,884,323 A		3/1999	Hawkins et al.	707/201
5,928,329 A	*	7/1999	Clark et al.	709/227
5,960,406 A	*	9/1999	Rasansky et al.	707/500
5,961,590 A	*	10/1999	Mendez et al.	709/206
6,016,478 A	*	1/2000	Zhang et al.	709/206
6,018,761 A	*	1/2000	Uomini	709/206
6,034,621 A	*	3/2000	Kaufman	370/310

OTHER PUBLICATIONS

O'Connor et al., "Managing Contacts in Windows 95", PC User, Apr. 1996.*
Microsoft Office 97/Visual Basic Programmer's Guide, Chapter 5 Microsoft Outlook Objects. Last updated Feb. 3, 1997.
The Microsoft Outlook 97 Automation Server Programming Model, Published Mar. 3, 1997 by Randy Byrne.

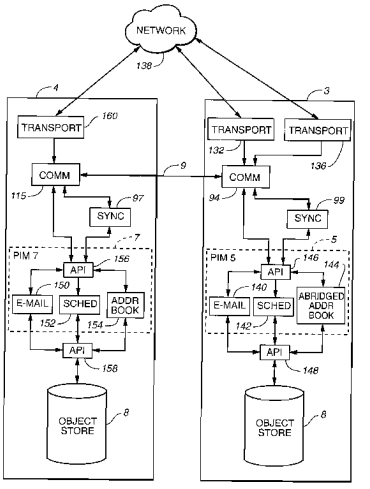
(List continued on next page.)

Primary Examiner—Ayaz Sheikh
Assistant Examiner—Philip B. Tran
(74) *Attorney, Agent, or Firm*—Joseph R. Kelly; Westman, Champlin & Kelly, P.A.

(57) **ABSTRACT**

The present invention includes a mobile device which provides the user with the ability to schedule a meeting request from the mobile device itself. The mobile device creates an object representative of the meeting request and assigns the object a global identification number which uniquely identifies the object to other devices which encounter the object. In addition, the mobile device in accordance with one aspect of the present invention provides a property in the object which is indicative of whether the meeting request has already been transmitted. In this way, other devices which encounter the meeting request are capable of identifying it as a unique meeting request, and of determining whether the meeting request has already been transmitted, in order to alleviate the problem of duplicate meeting request transmissions.

23 Claims, 8 Drawing Sheets



OTHER PUBLICATIONS

RecurringEvent Object. 1998 Microsoft Corporation.
RecurrencePattern Object. 1998 Microsoft Corporation.
AppointmentItem Object. 1998 Microsoft Corporation.
"Method for Personal Digital Assistance Calendar Export Nomenclature" for *IBM® Technical Disclosure Bulletin*, vol. 37 No. 3, Mar. 1994.
"The CallManager system: A platform for intelligent telecommunications services", by David J. Pepper, Sharad Singhal and Scott Soper, for *Speech Communication*, vol. 23, 1997.

* cited by examiner

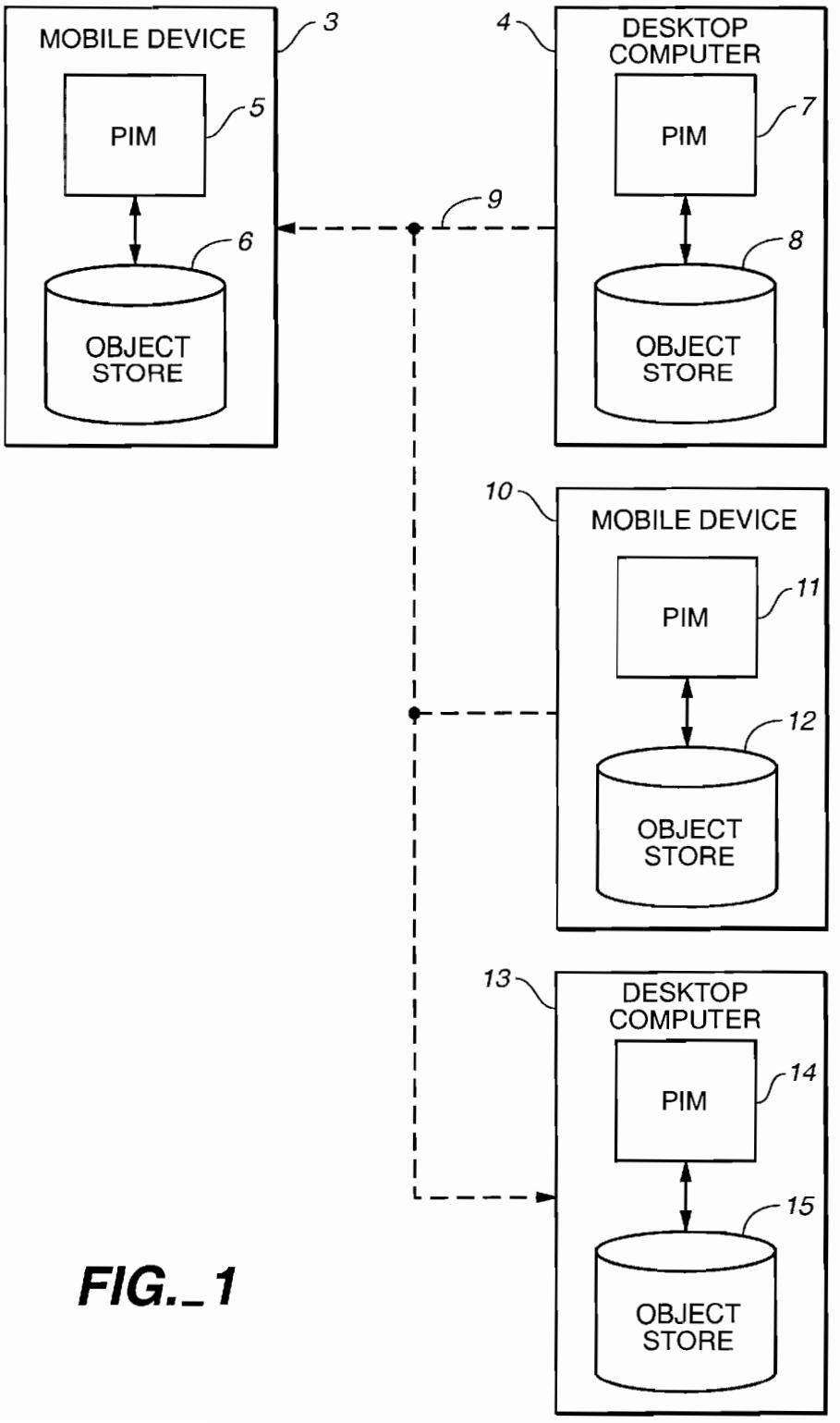

MOBILE DEVICE
3
PIM
5
6
OBJECT STORE
4
DESKTOP COMPUTER
PIM
7
8
OBJECT STORE
9
10
MOBILE DEVICE
PIM
11
12
OBJECT STORE
13
DESKTOP COMPUTER
PIM
14
15
OBJECT STORE

FIG._1

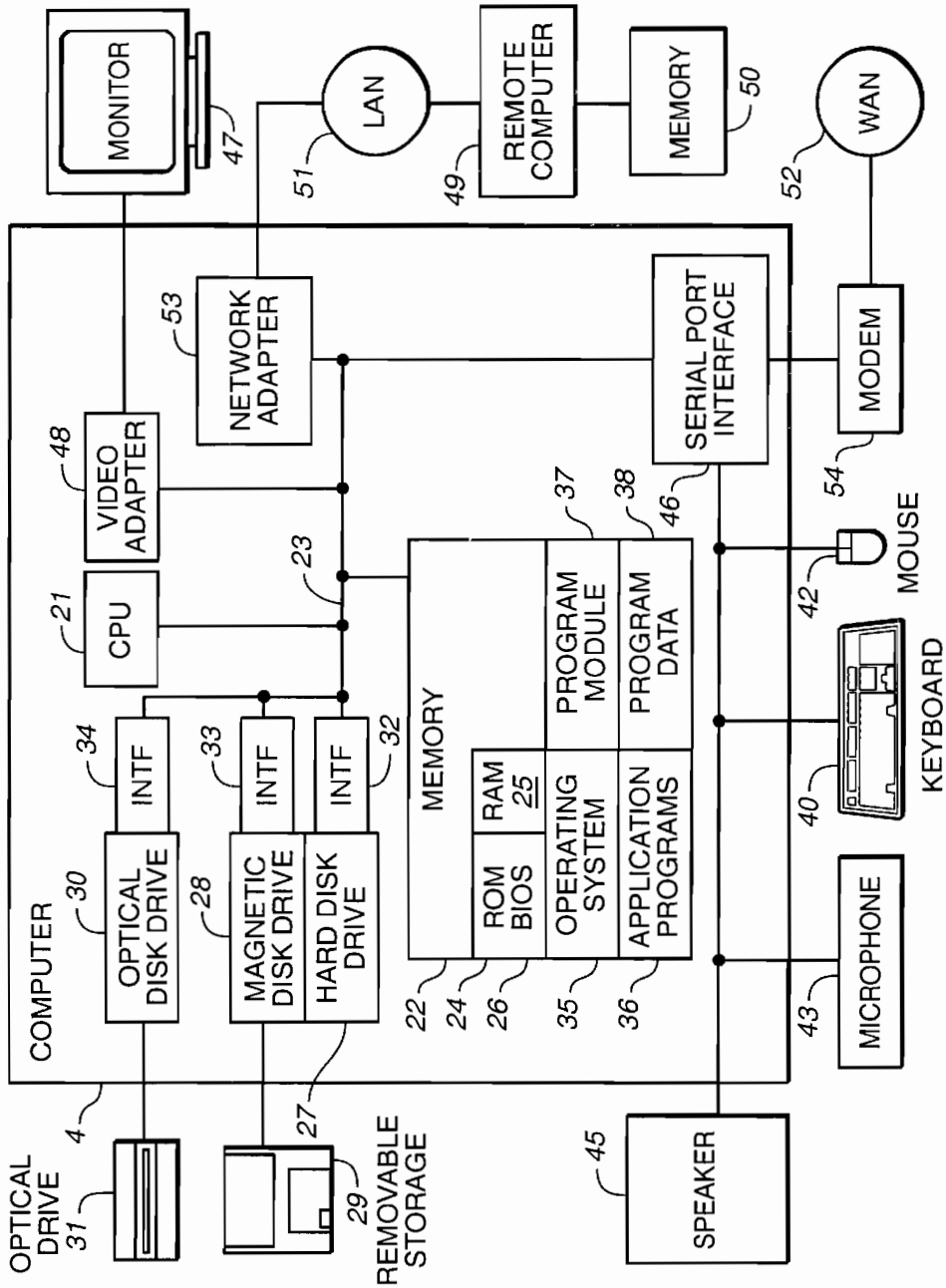

MONITOR
47
LAN
51
REMOTE COMPUTER
49
MEMORY
50
WAN
52
NETWORK ADAPTER
53
SERIAL PORT INTERFACE
MODEM
54
VIDEO ADAPTER
48
CPU
21
23
MEMORY
ROM BIOS
RAM 25
OPERATING SYSTEM
APPLICATION PROGRAMS
PROGRAM MODULE
PROGRAM DATA
37
38
46
MOUSE
42
KEYBOARD
40
MICROPHONE
43
SPEAKER
45
INTF
34
INTF
33
INTF
32
OPTICAL DISK DRIVE
30
MAGNETIC DISK DRIVE
28
HARD DISK DRIVE
27
REMOVABLE STORAGE
29
OPTICAL DRIVE
31
4
COMPUTER
22
24
26
35
36

FIG._2

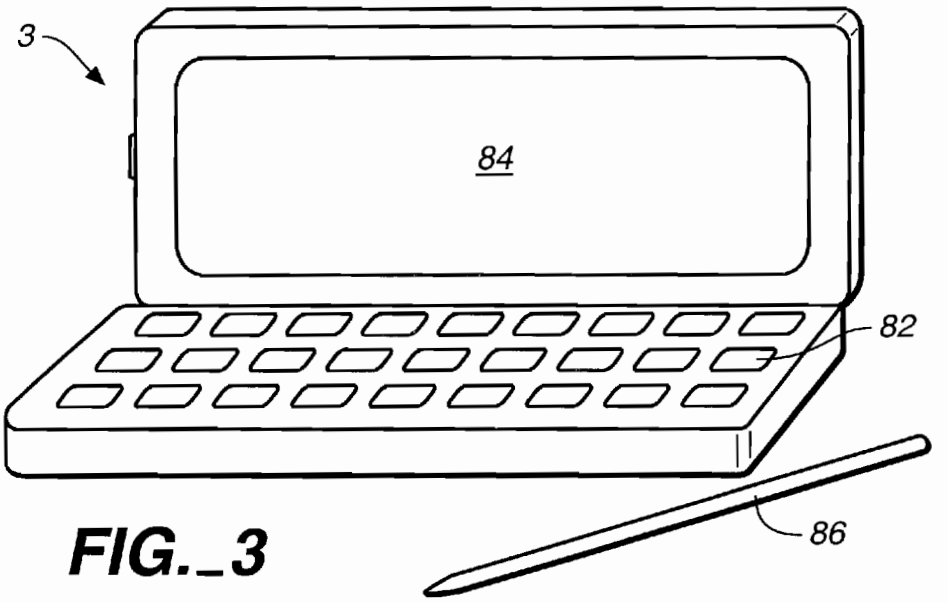

3
84
82
86

FIG._3

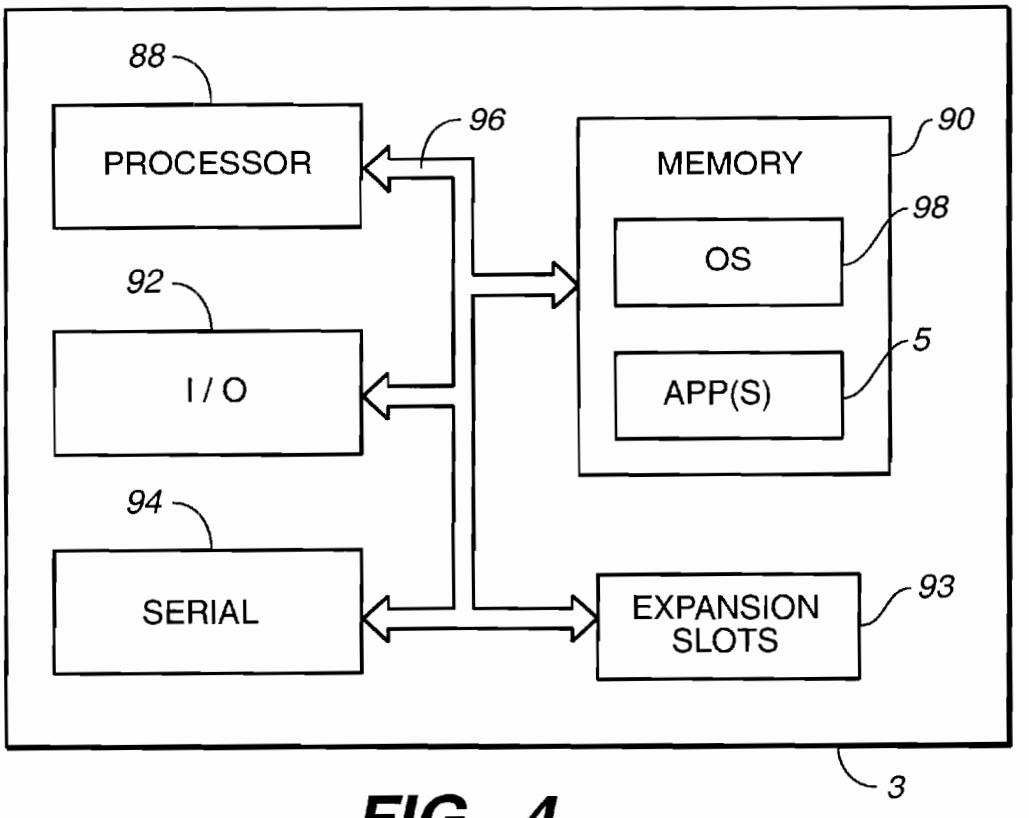

88
PROCESSOR
96
90
MEMORY
98
OS
92
I / O
5
APP(S)
94
SERIAL
EXPANSION SLOTS
93
3

FIG._4

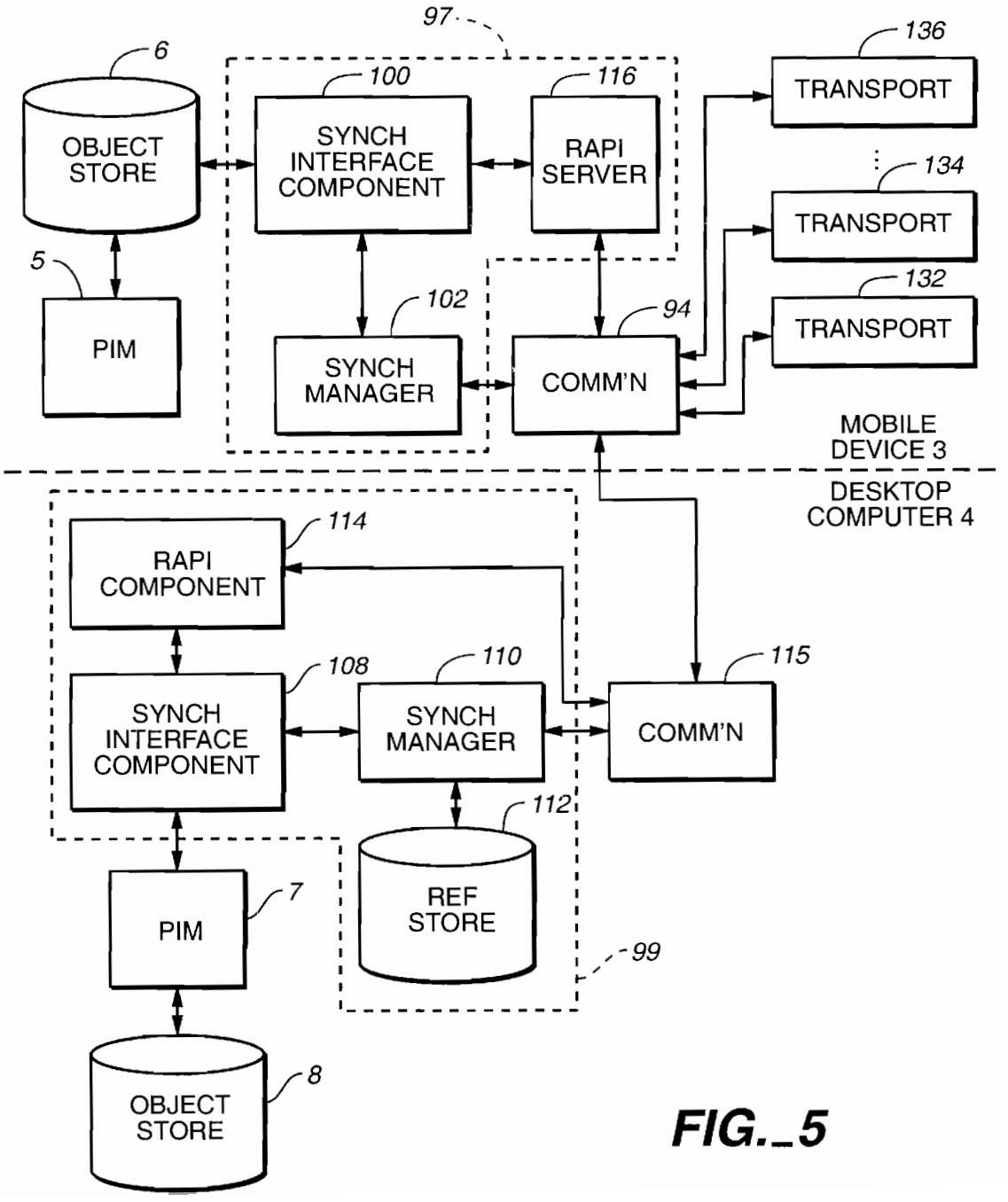

97
6
OBJECT STORE
100
SYNCH INTERFACE COMPONENT
116
RAPI SERVER
136
TRANSPORT
134
TRANSPORT
132
TRANSPORT
5
PIM
102
SYNCH MANAGER
94
COMM'N
MOBILE DEVICE 3
DESKTOP COMPUTER 4
114
RAPI COMPONENT
108
SYNCH INTERFACE COMPONENT
110
SYNCH MANAGER
115
COMM'N
112
REF STORE
7
PIM
99
8
OBJECT STORE

FIG._5

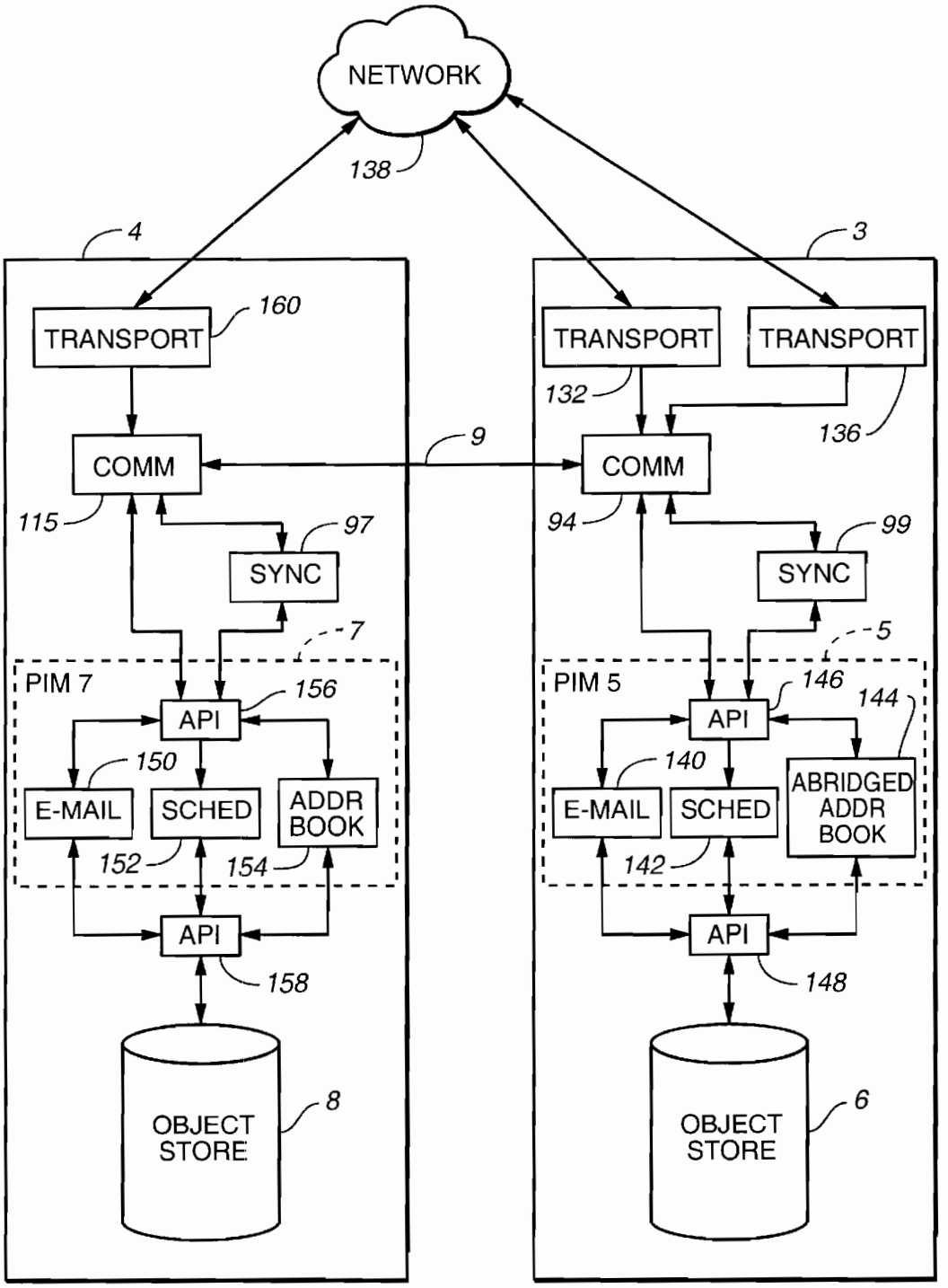

NETWORK
138
4
3
TRANSPORT
160
TRANSPORT
TRANSPORT
132
136
COMM
9
COMM
115
94
97
99
SYNC
SYNC
7
5
PIM 7
PIM 5
API
156
146
144
API
150
140
E-MAIL
SCHED
ADDR BOOK
E-MAIL
SCHED
ABRIDGED ADDR BOOK
152
154
142
API
API
158
148
OBJECT STORE
8
OBJECT STORE
6

FIG._6

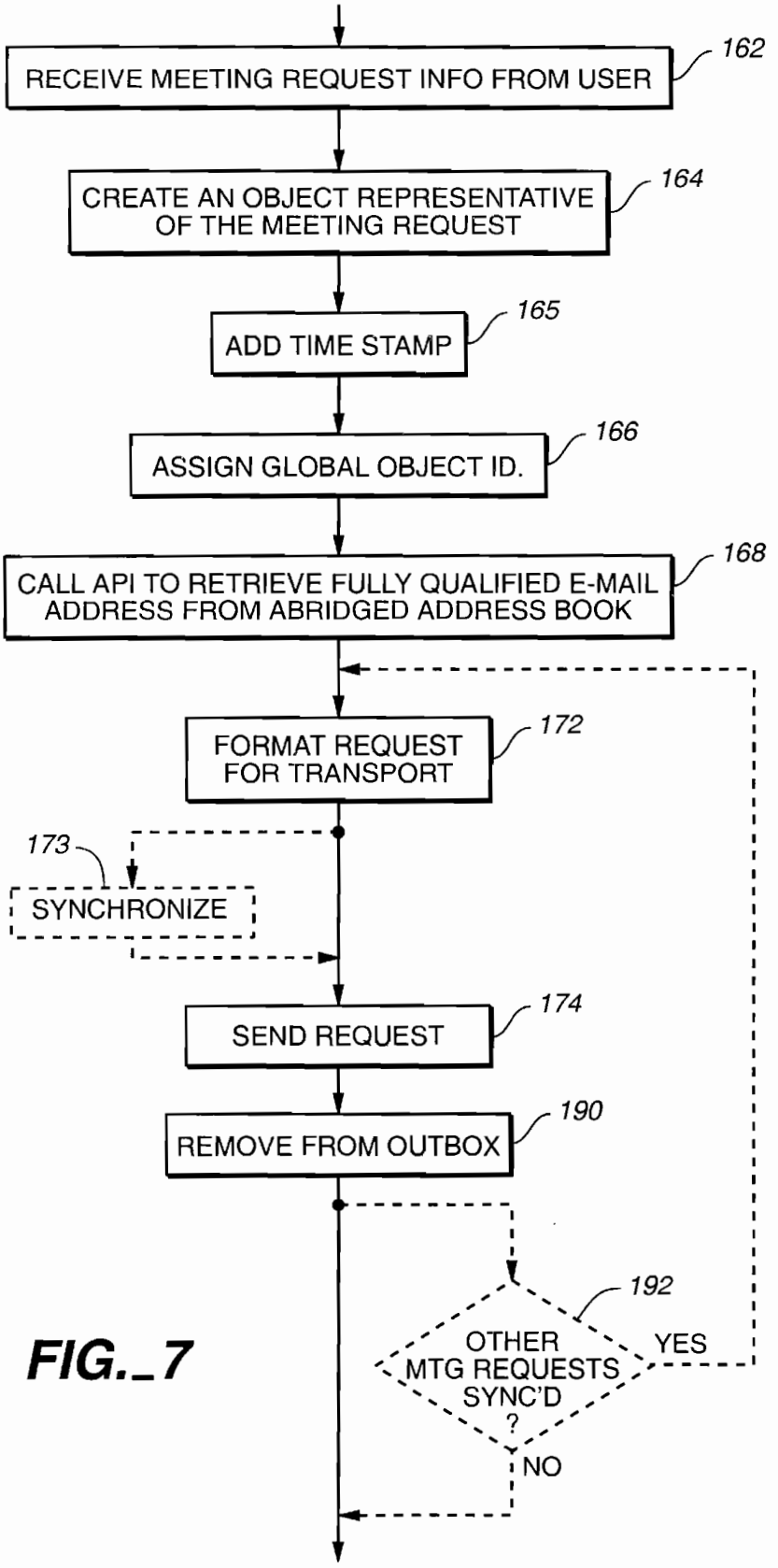

RECEIVE MEETING REQUEST INFO FROM USER
162
CREATE AN OBJECT REPRESENTATIVE OF THE MEETING REQUEST
164
ADD TIME STAMP
165
ASSIGN GLOBAL OBJECT ID.
166
CALL API TO RETRIEVE FULLY QUALIFIED E-MAIL ADDRESS FROM ABRIDGED ADDRESS BOOK
168
FORMAT REQUEST FOR TRANSPORT
172
173
SYNCHRONIZE
SEND REQUEST
174
REMOVE FROM OUTBOX
190
OTHER MTG REQUESTS SYNC'D ?
192
YES
NO

FIG._7

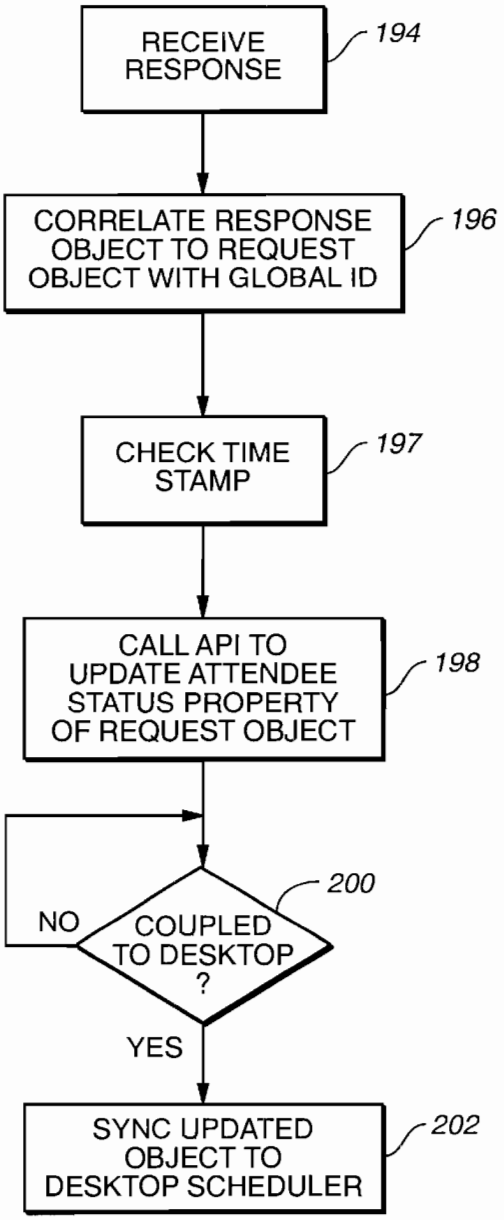

RECEIVE RESPONSE
194
CORRELATE RESPONSE OBJECT TO REQUEST OBJECT WITH GLOBAL ID
196
CHECK TIME STAMP
197
CALL API TO UPDATE ATTENDEE STATUS PROPERTY OF REQUEST OBJECT
198
NO
COUPLED TO DESKTOP ?
200
YES
SYNC UPDATED OBJECT TO DESKTOP SCHEDULER
202

FIG._8

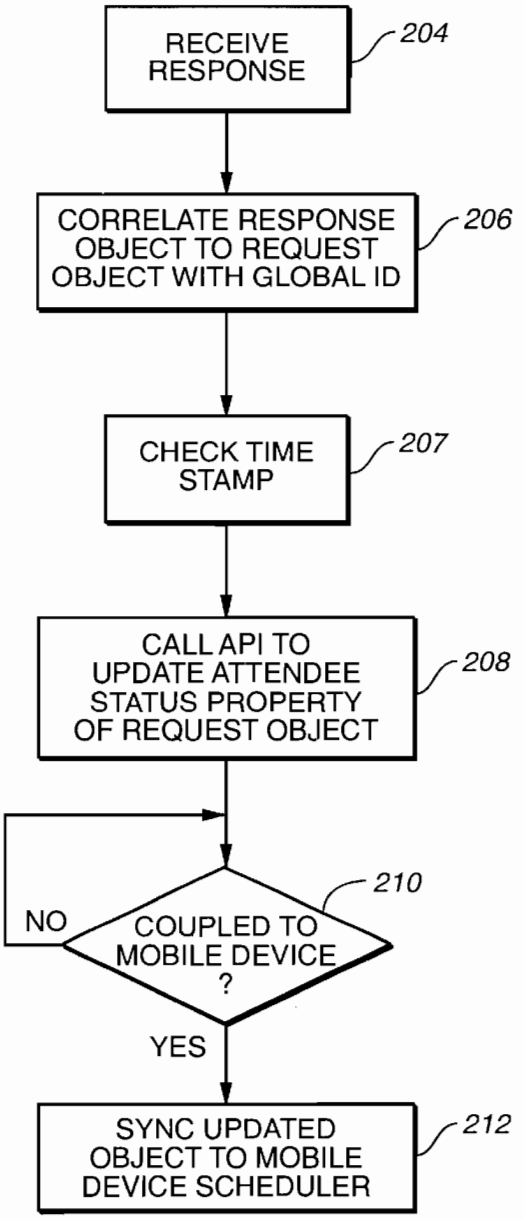

RECEIVE RESPONSE
204
CORRELATE RESPONSE OBJECT TO REQUEST OBJECT WITH GLOBAL ID
206
CHECK TIME STAMP
207
CALL API TO UPDATE ATTENDEE STATUS PROPERTY OF REQUEST OBJECT
208
NO
COUPLED TO MOBILE DEVICE ?
210
YES
SYNC UPDATED OBJECT TO MOBILE DEVICE SCHEDULER
212

FIG._9

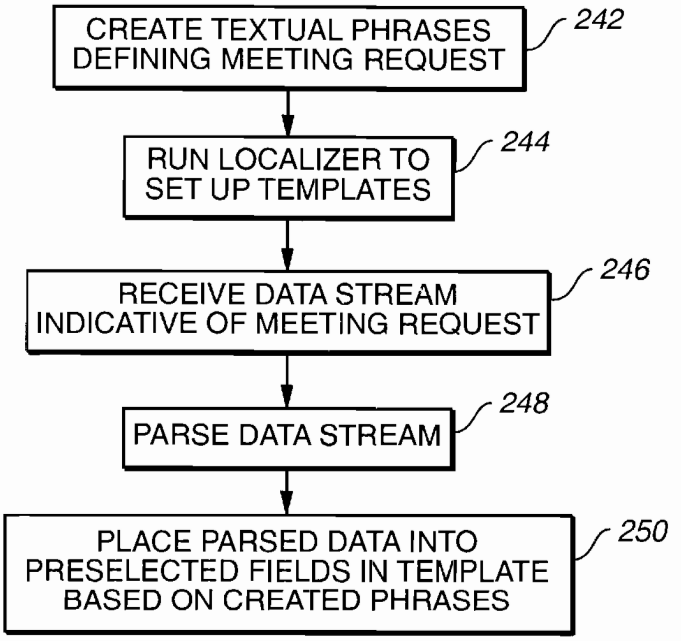


FIG._10

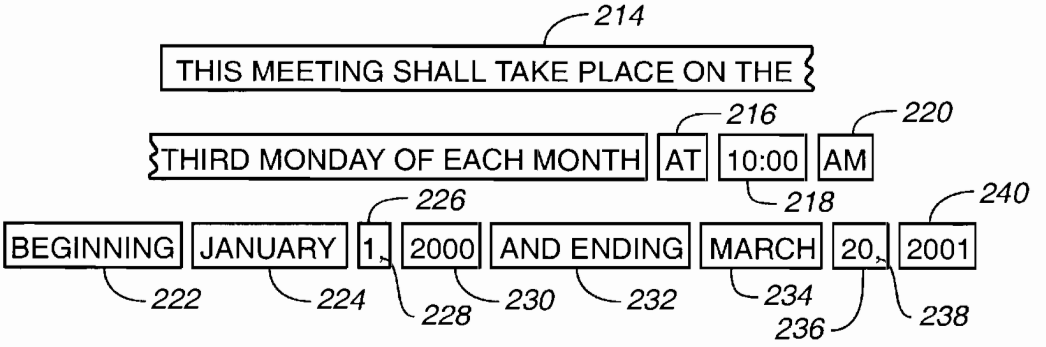


FIG._11A

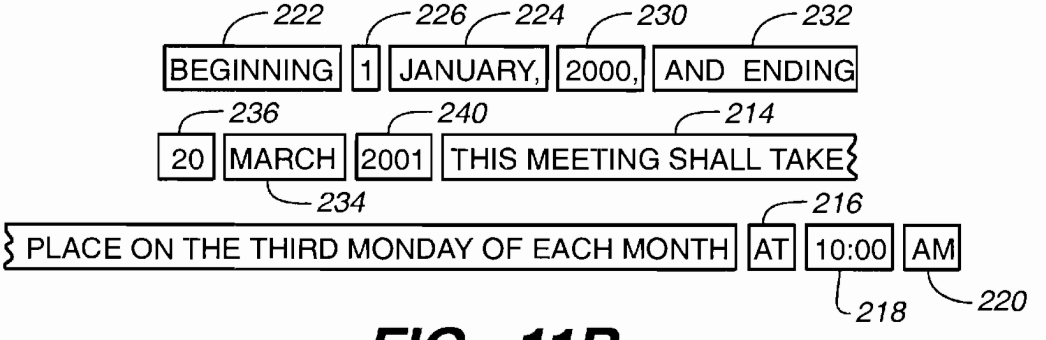


FIG._11B

GENERATING MEETING REQUESTS AND GROUP SCHEDULING FROM A MOBILE DEVICE

REFERENCE TO CO-PENDING PATENT APPLICATION

This patent application claims the priority of the following two U.S. provisional patent applications, Ser. No. 60/063,164 entitled "FEATURES OF A MOBILE DEVICE AND ASSOCIATED COMPUTER", filed on Oct. 24, 1997 and Ser. No. 60/064,986 entitled "FEATURES OF A MOBILE DEVICE AND ASSOCIATED COMPUTER", filed on Nov. 7, 1997.

Reference is hereby made to the following co-pending U.S. patent applications which are hereby incorporated by reference:

Ser. No. 09/058,613, filed on even date herewith, entitled "ELECTRONIC MAIL OBJECT SYNCHRONIZATION BETWEEN A DESKTOP COMPUTER AND MOBILE DEVICE"; and

Ser. No. 09/058,528, filed on even date herewith, entitled "SYSTEM AND METHOD FOR INTERACTION BETWEEN A DESKTOP COMPUTER AND MULTIPLE MOBILE DEVICE", all of which are assigned to same assignee as the present invention.

BACKGROUND OF THE INVENTION

Mobile devices are small electronic computing devices often referred to as personal digital assistants. One such mobile device is sold under the trade name Handheld PC (or "H/PC") based on a Windows CE brand operating system provided by Microsoft Corporation of Redmond, Wash. While a wide variety of computing tasks and applications can be performed by such mobile devices, personal information managers (PIMs) are particularly well suited to mobile devices.

PIMs typically comprise applications which enable the user of the mobile device to better manage scheduling and communications, and other such tasks. Some commonly available PIMs include scheduling and calendar programs, task lists, address books, and electronic mail (e-mail) programs. Some commonly commercially available PIMs are sold under the brand names Microsoft Schedule+ and Microsoft Outlook and are commercially available from Microsoft Corporation of Redmond, Wash. For purposes of this discussion, PIMs shall also include separate electronic mail applications, such as that available under the brand name Microsoft Exchange.

It is also common for mobile devices to be used in conjunction with a desktop computer. For example, the user of a mobile device may also have access to, and use, a desktop computer at work, at home, or both. A user may typically run the same types of PIMs on both the desktop computer and the mobile device (although the particular versions of the PIMs may be somewhat different from the desktop computer to the mobile device). Thus, it is quite advantageous for the mobile device to be designed to be couplable to the desktop computer to exchange information with, and share information with, the desktop computer.

The user may also typically make changes to the PIMs both on the mobile device, and at the desktop. Therefore, it is advantageous for the PIMs on both the mobile device and the desktop to contain the most up-to-date information, regardless of whether recent changes to the PIMs have been made on the mobile device or the desktop computer. The process of coupling the mobile device with the desktop computer, and integrating the information stored by the PIMs on the mobile device and the desktop computer such that the two contain the same updated information is referred to as synchronization.

Conventional PIMs which support electronic calendaring and scheduling features (collectively referred to as a scheduler, or as a scheduling application) are traditionally supported on desktop computers. Such PIMs provide the ability of the user to schedule a meeting request for one or more desired attendees.

In order to generate a meeting request, the user typically interacts with the scheduling application through a user interface. The user interface provides the user with a plurality of selectable options to parameterize the meeting request. For example, the user interface typically allows the user to pick a date and time (and often a place) on which the meeting is to be held. The user interface also typically allows the user to select a group of attendees that the user wishes to attend the meeting, to enter some textual description of the meeting, and to specify whether the meeting is for only a single date, or is a recurring meeting (i.e., whether the meeting is to occur only on one date, the 15^{th} of every month, the first Monday of every month, every Monday, etc.).

Based on this information, the scheduling application creates an object which is representative of the meeting and enters it on the user's calendar as an appointment. Such objects are typically defined by a number of properties, some of which are defined by the user input information which the user provides while generating the meeting request. The meeting object also contains a critical time stamp (UTC) which is updated whenever a critical change is made to the meeting object, such as changes to the start or end date or time, changes in the location, etc.

Since other people are identified as attendees, the appointment entered on the calendar is viewed as a meeting and the scheduling application typically calls methods exposed by an electronic mail application in accordance with messaging application programming interfaces (MAPI), or other APIs which are a set of well documented, published interfaces commercially available from the Microsoft Corporation of Redmond, Wash.

In response, the electronic mail application creates another object (an electronic mail meeting request object) indicative of the meeting request and the electronic mail application (or suitable transport) formats this electronic mail meeting request object into a well defined electronic mail message suitable for transmission. In doing so, the critical time stamp from the meeting object is also placed in the electronic mail meeting request object. The electronic mail application then interacts with a specified transport and transports the electronic mail meeting request object to a network which routes it to the designated attendees. In doing so, the electronic mail application typically accesses an address book stored in a database to obtain the fully qualified electronic mail address for the attendees. This is also typically done by calling MAPI or other suitable API methods associated with the database storing the address book. The generation of the meeting object and the creation of the electronic mail meeting request object will be referred to herein collectively as creating a meeting request.

The potential attendees then typically respond to the meeting request. In doing so, the originator's critical time stamp is sent back (unmodified) along with the response. The response also includes a recipient critical time stamp

and an indication of the recipient's response (e.g., accept, decline, tentative, etc.). The recipient critical time stamp is updated by the recipient (potential attendee) whenever a critical change is made by the recipient. This allows the user to reliably order receipt of multiple versions of the same meeting (e.g., where the originator changes the time, date or location of the meeting such that multiple meeting requests are generated). It also allows the originator to reliably order receipt of responses and ensure that each response correlates to the most recent version of the meeting.

The response is then transmitted back to the originator (e.g., the sending computer). The electronic mail application and scheduling application on the originator then typically process the response (or responses) accordingly. For example, the originator stores, for each recipient (or potential attendee) the recipient critical time stamp in a table along with each recipient's response code (which is indicative of the accept, decline, tentative response). The two commercially available PIMs identified above (the Microsoft Schedule+ and Microsoft Outlook brand PIMs) are examples of PIMs which support the features discussed above.

Meeting cancellations, and exceptions to recurring meeting must also be handled. For example, the PIMs may allow scheduled meetings to be cancelled, and allow a variety of exceptions to a recurring meeting pattern.

Scheduling of meeting requests as described above has, to date, only been supported by desktop computers or laptop computers which are fitted with a hard disk drive or other high capacity memory mechanisms, or by low intelligence terminals which are permanently attached to a server or other similar computer which, itself, contains a high capacity storage device. The ability to schedule a meeting request from a mobile device is simply unavailable. While some current mobile devices are provided with PIMs that allow the user to view meeting requests, and to view meetings which have already been scheduled, current mobile devices do not allow the user to generate a meeting request from the mobile device itself.

A number of significant obstacles present themselves when attempting to provide the user with the capability of generating a meeting request from a mobile device. Meeting cancellations and exceptions to recurring meetings must be handled. Also, a significant problem arises with respect to the possibility of transmitting duplicate meeting requests. While duplicate meeting requests as described below may not necessarily be created with all PIMs, they do present a potential problem which must be considered. For example, if the user of the mobile device were able to generate a meeting request, a meeting object would first be entered on the calendar of the mobile device. The electronic mail application on the mobile device would then create a corresponding electronic mail meeting request object. The next time the mobile device was synchronized with the desktop computer, the meeting object would be synchronized with the calendar object store on the desktop computer and the electronic mail meeting request object would be synchronized to the desktop outbox. The desktop computer, would recognize the electronic mail meeting request object in its outbox, format it for transmission, and transmit it over the network. Further, synchronizing the meeting object to the calendar of the desktop computer may result in another electronic mail meeting request object being created and transmitted by the desktop computer. This would result in duplicate electronic mail meeting request objects being created (one by the mobile device, and one by the desktop computer after synchronization) and transmitted. Under that scenario, potential attendees would receive two or more meeting requests, and may respond to both. This would create duplicate responses to what was intended to be a single meeting request.

A similar problem may occur if a meeting request were generated in a conventional manner (on a desktop computer) for instance, and was then synchronized to a mobile device having the capability of generating and transmitting meeting requests. The meeting object on the desktop calendar would be synchronized to the calendar of the mobile device. The mobile device might then recognize the meeting object synchronized from the desktop, and create an electronic mail meeting request object and attempt to transmit the object. This would result in substantially the same problem—duplicate meeting requests and duplicate responses for what was intended to be a single meeting request.

Further, if the user of the mobile device coupled the mobile device for synchronization with more than one desktop computer (e.g., a home computer and work computer, if the mobile device were provided with this capability) the same problem would result. In that instance, and using conventional architecture, both desktop computers would synchronize with, and recognize, the meeting object and the electronic mail meeting request object from the mobile device. The desktop computers would both potentially create additional electronic mail meeting request objects and transmit them to the potential attendees. Again, this would result in many different meeting requests and responses being transmitted for what was intended to be only a single meeting request.

In addition, if mobile devices were provided with the capability of being connected directly to one another, and communicating with one another, without going through a desktop or similar computer, the meeting request could be generated by one mobile device, responded to by another mobile device, and scheduled on both mobile devices. However, the next time the first mobile device is synchronized with the desktop computer, that computer might again recognize the meeting object synchronized from the calendar of the mobile device, create another electronic mail meeting request object and transmit the electronic mail meeting request object. Thus, a significant problem potentially exists with respect to the generation of multiple meeting requests.

Of course, similar problems also result from critical changes to the meeting object on either the desktop computer or the mobile device. This will cause unwanted duplicate electronic mail meeting requests in a similar fashion.

Additional problems also present themselves simply by the fact that conventional mobile devices have a memory capacity which is significantly less than that of a desktop computer or similar computer. Thus, problems arise with respect to storing address books on the mobile device itself which contain the fully qualified electronic mail address of all potential attendees.

Further obstacles present themselves because many desktop computers on which the meeting request must be processed have different scheduling applications. Therefore, the meeting request generated by the mobile device may be incompatible with scheduling applications which it encounters.

In addition, localization of meeting requests can present a problem. For instance, in some localities, it is conventional, when writing a date, to place the month first, the day second and the year third. In other localities, other orders are conventional. Further, a textual description which describes

the meeting and which accompanies the meeting request, may need to be rearranged to conform to local convention. Also, meeting requests can be generated in one time zone and transmitted to recipients in other time zones. This can tend to be confusing.

The present invention addresses some or all of these obstacles.

SUMMARY OF THE INVENTION

The present invention includes a mobile device which provides the user with the ability to schedule a meeting request from the mobile device itself. The mobile device creates an object representative of the meeting request and assigns the object a global identification number which uniquely identifies the object to other devices which encounter the object. In this way, other devices which encounter the meeting request are capable of identifying it as a unique meeting request in order to alleviate the problem of duplicate meeting request transmissions.

In accordance with another preferred feature of the present invention, an electronic mail application or calendar application on the mobile device obtains a fully qualified electronic mail address for the potential attendees from an abridged address book or directory stored on the mobile device itself. This alleviates problems associated with the storage capacity of the mobile device.

In accordance with another preferred embodiment of the present invention, the mobile device creates the meeting object and the electronic mail meeting request object using a set of properties which are supported by a plurality of PIMs that may receive the objects. This provides compatibility with an increased number of devices which are likely to encounter the objects.

In accordance with yet another preferred feature of the present invention, localizers implement a plurality of templates on the mobile device which are used in formatting the properties of the objects associated with the meeting request. A data stream representative of the meeting request is parsed by the mobile device and placed in pre-defined fields in the appropriate templates so that the text viewed by the user of the mobile device more closely conforms to local convention. In addition, time zone information is also included in one embodiment.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram illustrating a basic environment of the present invention.

FIG. **2** is a block diagram of one embodiment of a conventional desktop computer used in conjunction with a mobile device in accordance with the present invention.

FIG. **3** is a simplified pictorial illustration of one embodiment of a mobile device in accordance with the present invention.

FIG. **4** is a simplified block diagram of one embodiment of the mobile device shown in FIG. **3**.

FIG. **5** is an architectural block diagram illustrating one embodiment of portions of the desktop computer shown in FIG. **2** and the mobile device shown in FIGS. **3** and **4** to illustrate synchronization of information stored in object stores on the desktop computer and the mobile device in accordance with the present invention.

FIG. **6** is an architectural block diagram illustrating one embodiment of portions of the desktop computer shown in FIG. **2** and the mobile device shown in FIGS. **3** and **4** to illustrate the generation and transmission of a meeting request.

FIG. **7** is a flow diagram illustrating the generation of a meeting request in accordance with one preferred embodiment of the present invention.

FIG. **8** is a flow diagram illustrating the handling of responses to a meeting request on the mobile device shown in FIGS. **3** and **4** in accordance with one preferred embodiment of the present invention.

FIG. **9** is a flow diagram illustrating the handling of responses to a meeting request on the desktop computer shown in FIG. **2** in accordance with one preferred embodiment of the present invention.

FIG. **10** is a flow diagram illustrating the use of a localizer to localize the textual description of a meeting request in accordance with one preferred embodiment of the present invention.

FIGS. **11**A and **11**B illustrate one embodiment of a template used to localize the textual description of a meeting request in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

OVERVIEW

FIG. **1** is a block diagram of a typical system or environment **2** in which the present invention operates. Environment **2** includes mobile device **3** and desktop computer **4**. FIG. **1** also illustrates that mobile device **3** can optionally be separately coupled to another mobile device **10** or another desktop computer **13**.

Mobile device **3** includes an application program **5** and an object store **6**. Desktop computer **4** also includes an application program **7** and an object store **8**. Mobile device **10** includes an application program **11** and object store **12**. Further, desktop computer **13** includes an application program **14** and an object store **15**.

Mobile device **3** is couplable to desktop computer **4**, mobile device **10** or desktop computer **13** by one of a plurality of connection mechanisms **9**. The operation of desktop computers **4** and **13** and the operation of mobile devices **3** and **10** are preferably similar. Therefore, for the sake of simplicity, the present description proceeds only with respect to mobile device **3** and desktop computer **4**.

In one preferred embodiment of the present invention, application program **7** on desktop **4** is a personal information manager (PIM) which supports electronic mail messaging, scheduling and calendaring and an address book containing contact information.

PIM applications are not always integrated. For instance, some scheduling programs do not contain electronic mail features, but interface with whatever electronic mail program is provided, and vice versa. Of course, PIM **7** (whether it be a single application, a single integrated application or multiple interfaced applications) can be configured to support a wide variety of other features, such as task lists and personalized address books, to name a few. However, for the sake of clarity, only features relating to electronic mail messaging, scheduling and calendar features, and address book features are discussed in detail with respect to the present invention.

Object store **8** is a memory which is configured to store a plurality of individual records or objects, each comprising a plurality of fields or properties related to the above-mentioned features. In one preferred embodiment, PIM **7** is a program, such as that available under the commercial designation Microsoft Outlook **97**, and object store **8** is

configured to store objects, each of which has a plurality of properties which can be associated with electronic mail messaging, scheduling and calendaring, and contact information.

For example, some properties included in an object associated with electronic mail messaging include the sender's name, the recipient's name, text messages, an indication of whether attachments are attached to any given electronic mail message, and possibly other similar properties. Also, some properties associated with scheduling and calendaring include critical time stamp information, date and time information, potential attendees, recurrence properties which describe recurrent meetings or meeting requests, and a textual description of the meeting request. Some properties associated with contact information include the proper names, or familiar names, of those in the contact list, the addresses and telephone numbers of those listed, and the fully qualified electronic mail address for those in the list. Any number of other properties can also be included. Desktop computer **4** executes the application program identified as PIM **7** to maintain objects stored in object store **8**.

The application program designated as PIM **5** for mobile device **3** is a similar PIM to that stored on desktop computer **4**. Object store **6** on mobile device **3** is configured to store a plurality of individual records or objects, each comprising a plurality of fields or properties related to the above-mentioned, supported features. As with PIM **7**, PIM **5** can also support a wide variety of other features. However, for the sake of clarity, only those features related to the present invention are discussed in detail herein.

In one illustrative embodiment, each object in object store **6** comprises the same set of properties stored in object store **8** for related messages, or a subset of those properties. In addition, the objects stored in object store **6** also preferably include a critical change time stamp, properties which indicate whether potential attendees identified in a meeting request have responded, and a global identification of each individual meeting request (discussed in greater detail below). Mobile device **3** executes PIM **5** to maintain the objects in object store **6**.

In the illustrative embodiment, each object stored in object store **8** is also stored in object store **6**. However, there are actually two instances of each object (one in object store **6** and one in object store **8**). Thus, when a user changes one instance of the object stored in either store **6** or store **8**, the second instance of that object in the other of stores **6** and **8** is preferably updated the next time mobile device **3** is connected to desktop computer **4** so that both instances of the same object contain up-to-date data. This is referred to as synchronization.

In order to accomplish synchronization, synchronization components run on both mobile device **3** and desktop computer **4**. The synchronization components communicate with PIMs **5** and **7** on mobile device **3** and desktop computer **4** through well defined interfaces to manage communication and synchronization.

The components of mobile device **3** and desktop computer **4** communicate with each other through any suitable, and commercially available, communication link **9**, and using a suitable communications protocol. For instance, in one preferred embodiment, mobile device **3** is connectable to desktop computer **4** with a physical cable which communicates using a serial communications protocol. Other communication mechanisms are also contemplated by the present invention, such as infrared (IR) communication, direct modem communication, remote dial-up networking communication, communication through commercially available network cards (i.e., using TCP/IP), remote access services (RAS), wireless modem, wireless cellular digital packet data (CDPD), or other suitable communication mechanisms.

FIG. **2** and the related discussion are intended to provide a brief, general description of a suitable desktop computer **4** in which portions of the invention may be implemented. Although not required, the invention will be described, at least in part, in the general context of computer-executable instructions, such as program modules, being executed by a personal computer **4** or mobile device **3**. Generally, program modules include routine programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that desktop computer **4** may be implemented with other computer system configurations, including multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices. Examples of distributed programs include those available under the commercial designations Exchange, Schedule+ and MS Mail, all available from Microsoft Corporation.

With reference to FIG. **2**, an exemplary system for implementing desktop computer **4** includes a general purpose computing device in the form of a conventional personal computer **4**, including processing unit **21**, a system memory **22**, and a system bus **23** that couples various system components including the system memory to the processing unit **21**. The system bus **23** may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) **24** and random access memory (RAM) **25**.

A basic input/output system (BIOS) **26**, containing the basic routine that helps to transfer information between elements within the desktop computer **4**, such as during start-up, is stored in EEPROM which is part of ROM **24**. The desktop computer **4** further preferably includes a hard disk drive **27** for reading from and writing to a hard disk (not shown), a magnetic disk drive **28** for reading from or writing to removable magnetic disk **29**, and an optical disk drive **30** for reading from or writing to a removable optical disk **31** such as a CD ROM or other optical media. The hard disk drive **27**, magnetic disk drive **28**, and optical disk drive **30** are connected to the system bus **23** by a hard disk drive interface **32**, magnetic disk drive interface **33**, and an optical drive interface **34**, respectively. The drives and the associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, program modules and other data for the desktop computer **4**.

Although the exemplary environment described herein employs a hard disk, a removable magnetic disk **29** and a removable optical disk **31**, it should be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks (DVDs), Bernoulli cartridges, random access memories (RAMs), read only memory (ROM), and the like, may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk, magnetic disk **29**, optical disk **31**, ROM **24** or RAM **25**,

including an operating system **35**, one or more application programs **36** (which include PIMs **7**), other program modules **37**, and program data **38**. A user may enter commands and information into the desktop computer **4** through input devices such as a keyboard **40**, pointing device **42** and microphone **62**. Other input devices (not shown) may include a joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit **21** through a serial port interface **46** that is coupled to the system bus **23**, but may be connected by other interfaces, such as a sound card, a parallel port, a game port or a universal serial bus (USB). A monitor **47** or other type of display device is also connected to the system bus **23** via an interface, such as a video adapter **48**. In addition to the monitor **47**, desktop computers may typically include other peripheral output devices such as speaker **45** and printers.

The desktop computer **4** may operate in a networked environment using logic connections to one or more remote computers (other than mobile device **3**), such as a remote computer **49**. The remote computer **49** may be another personal computer, a server, a router, a network PC, a peer device or other network node, and typically includes many or all of the elements described above relative to desktop computer **4**, although only a memory storage device **50** has been illustrated in FIG. **2**. The logic connections depicted in FIG. **2** include a local area network (LAN) **51** and a wide area network (WAN) **52**. Such networking environments are commonplace in offices, enterprise-wide computer network intranets and the Internet.

When used in a LAN networking environment, the desktop computer **4** is connected to the local area network **51** through a network interface or adapter **53**. When used in a WAN networking environment, the desktop computer **4** typically includes a modem **54** or other means for establishing communications over the wide area network **52**, such as the Internet. The modem **54**, which may be internal or external, is connected to the system bus **23** via the serial port interface **46**. In a network environment, program modules depicted relative to desktop computer **4**, or portions thereof, may be stored in the remote memory storage devices. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Desktop computer **4** runs operating system **35** that is typically stored in non-volatile memory **24** and executes on the processor **21**. One suitable operating system is a Windows brand operating system sold by Microsoft Corporation, such as Windows 95 or Windows NT, operating systems, other derivative versions of Windows brand operating systems, or another suitable operating system. Other suitable operating systems include systems such as the Macintosh OS sold from Apple Corporation, and the OS/2 Presentation Manager sold by International Business Machines (IBM) of Armonk, N.Y. PIM **7** is preferably stored in program module **37**, in volatile memory or non-volatile memory, or can be loaded into any of the components shown in FIG. **2** from a floppy diskette **29**, CDROM drive **31**, downloaded from a network via network adapter **53**, or loaded using another suitable mechanism.

A dynamically linked library (DLL), comprising a plurality of executable functions is associated with PIM **7** for execution by processor **21**. Interprocessor and intercomponent calls are facilitated using the component object model (COM) as is common in programs written for Microsoft Windows brand operating systems. Briefly, when using COM, a software component such as a DLL has a number of interfaces. Each interface exposes a plurality of methods, which can be called individually to utilize different services offered by the software component. In addition, interfaces are provided such that methods or functions can be called from other software components which optionally receive and return one or more parameter arguments.

In general, the DLL associated with PIM **7** is designed specifically to work in conjunction with PIM **7** and to expose desktop synchronization interfaces that function as described in more detail in the above-referenced co-pending U.S. patent application according to a synchronization protocol. The DLL, in turn, calls interfaces exposed by PIM **7** in order to access data representing individual properties of objects maintained in object store **8**. Object store **8**, of course, can reside in any one of the suitable memory components described with respect to FIG. **2**.

FIG. **3** is a pictorial illustration of one preferred embodiment of a mobile device **3** which can be used in accordance with the present invention. Mobile device **3**, in one preferred embodiment, is a desktop assistant sold under the designation H/PC and being based on, for example, the Windows CE brand operating system provided by Microsoft Corporation. In one preferred embodiment, mobile device **3** is housed in a housing which fits comfortably in the hand of a typical user. Mobile device **3** has some components which are similar to those of desktop computer **4**. For instance, in one preferred embodiment, mobile device **3** includes a miniaturized keyboard **82**, display **84**, and stylus **86**. Of course, other configurations, such as without keyboards, can also be used.

In the embodiment shown in FIG. **3**, display **84** is a liquid crystal display (LCD) which uses a contact-sensitive display screen in conjunction with stylus **86**. Stylus **86** is used to press or contact the display **84** at designated coordinates to accomplish certain user input functions. Of course, other user input configurations can be used as well. For example, user input mechanisms could be included such as a keypad, a track ball, and other various types of miniaturized keyboards, or the like. In addition, mobile device **3** may not be embodied as the H/PC brand of digital assistant, but could also be implemented as another type of personal digital assistant (PDA), another personal organizer, a palm top computer, or a similar computerized notepad device.

FIG. **4** is a more detailed block diagram of mobile device **3**. Mobile device **3** preferably includes microprocessor **88**, memory **90**, input/output (I/O) components **92** (which include keyboard **82** and touch sensitive screen **84**), optional expansion slots **93** (which can be used for plugging in such things as PC cards, compact flash memory, etc.) and serial interface **94**. In the illustrative embodiment, these components of mobile device **3** are coupled for communication with one another over a suitable bus **96**.

In the illustrative embodiment, memory **90** is implemented as non-volatile electronic memory such as a random access memory (RAM) with a battery back-up module (not shown) such that information stored in memory **90** is not lost when the general power to mobile device **30** is shut down. A portion of memory **90** is preferably allocated as addressable memory for program execution, while another portion of memory **90** is preferably used to simulate storage on a disk drive.

Memory **90** includes operating system **98** and application (or PIM) **5**. Operating system **98**, during operation, is preferably loaded into, and executed by, processor **88** from memory **90**. Operating system **98**, in one preferred embodiment, is the Windows CE brand operating system. The operating system **98** is preferably designed for mobile

devices, and implements database features which can be utilized by PIM **5** through a set of exposed application programming interfaces and methods. The objects in object store **6** are preferably maintained by PIM **5** and operating system **98**, at least partially in response to calls to the exposed application program interfaces and methods.

It should be noted that PIM **5** is not necessarily designed to be entirely compatible with PIM **7** which executes on desktop computer **4**. For instance, there may not be a precise one-to-one matching between the properties of specific object types. Some properties supported by the electronic mail messaging features of PIM **5** may have no corresponding features in PIM **7** on desktop computer **4**, and vise versa.

OBJECT SYNCHRONIZATION

While object synchronization is described in greater detail in the above-referenced co-pending patent applications, it is described here, briefly, to assist in understanding the present invention.

FIG. **5** is an architectural block diagram illustrating one preferred embodiment of architectural components of mobile device **3** and desktop computer **4** which are used in synchronizing objects stored in object store **6** on mobile device **3** and object store **8** on desktop computer **4**. It should be noted that the synchronization architecture illustrated in FIG. **5** is but one preferred embodiment and others could be used as well. Further, the synchronization operation can be carried out between any two coupled stores (such as those on any combination of pairs of computers **4** and **13**, and mobile devices **3** and **10**). All that is required is a synchronization manager component, as described below, which directs synchronization of the two coupled stores. The synchronization manager can reside on either device, or both, or on a third computer coupled to the two stores. However, for the sake of simplicity, only the embodiment in which desktop computer **4** is coupled to mobile device **3** is described herein. Also, the present synchronization protocol is but one preferred embodiment, as another suitable. protocol could be implemented on top of standard electronic mail protocols.

In addition to PIM **5** and object store **6**, mobile device **3** includes synchronization module **97** which, in turn, includes interface component **100**, and synchronization manager **102**. Mobile device **3** also includes communications component **94**, remote application programming interface (RAPI) server **116**, and electronic mail messaging transports **132**, **134** and **136**.

Desktop computer **4** includes, in addition to PIM **7** and object store **8**, synchronization module **99** which, in turn, includes interface component **108**, synchronization manager **110**, reference store **112**, and RAPI component **114**. Desktop computer **4** also includes communications component **115**.

Generally, in the embodiment illustrated in FIG. **5**, synchronization manager **110** executes on desktop computer **4** and orchestrates synchronization between objects in object store **6** in handheld device **3**, and objects in object store **8** in desktop computer **4**. Synchronization manager **110** also maintains reference store **112** apart from desktop object store **8** as is described in greater detail below. Synchronization manager **110** implements the synchronization protocol to allow a comparison between corresponding objects stored in object store **6** in mobile device **3** and object store **8** in desktop computer **4**, to receive objects from object store **8**, and to update objects in object store **8**. The synchronization protocol also facilitates the retrieval of objects stored in object store **6** in mobile device **3** through synchronization interface component **100** and synchronization manager **102**, as well as communications component **94**.

On the side of mobile device **3**, the synchronization interface component **100** exposes application programming interfaces which synchronization manager **102** calls to read and store objects and object properties on object store **6**. In general, the application programming interfaces allow the creation of databases for different types of objects, and allow application programs to write and read property names and values to and from respective objects within object store **6**.

As discussed with respect to FIG. **1**, PIM **5** executes on mobile device **3** and maintains object store **6**. PIM **7** executes on desktop computer **4** and maintains object store **8**. There are many different ways which PIMs **5** and **7** can store objects in object stores **6** and **8**. However, in a preferred embodiment PIMs **5** and **7** create a distinct database for each object type. For example, different databases are created for meetings, contacts, tasks, electronic mail messages, etc. A predefined set of properties is supported for each object type, and each of the databases is assigned a name by the application program that creates it.

In an alternative embodiment, the application programs in PIMs **5** and **7** may use a single database for all object types, with the first property of each object defining the type of object. In any case, objects are uniquely identified within mobile device **3** and desktop computer **4** by object identifiers which are independent of the names assigned by the application programs creating the object.

Synchronization manager **110** is not necessarily closely associated with PIM **7**. Rather, it is an independent component which synchronizes objects from any application program that supports the appropriate desktop synchronization interfaces. The specific synchronization interfaces are described in greater detail in the co-pending patent application incorporated above. Communication components **94** and **115** implement serial communications between the computers using suitable link **9**.

Synchronization manager **110** communicates with PIM **7** and accesses object store **8** through synchronization interface component **108**. Synchronization interface component **108** corresponds generally to a DLL discussed with respect to FIG. **2** above, and exposes one or more application program interfaces and methods.

The interfaces and methods are described in greater detail in the co-pending patent application which is incorporated above by reference. Of these interfaces, one of note is preferably of the form known as messaging application program interfaces (MAPI) developed and published by the Microsoft Corporation for Windows brand operating system platforms, but other suitable interfaces can be used as well. In one preferred embodiment, the MAPI exposed by component **108** is a C-language application programming interface which allows programmable access to features of an electronic mail messaging program known as MS Mail also commercially available from the Microsoft Corporation. In another preferred embodiment, the MAPI exposed by component **108** is a component object model based (COM-based) set of interfaces which is sometimes referred to as extended MAPI and includes a set of automation interfaces to messaging systems, for use in Visual Basic and the like. However, synchronization interface component **108** and the associated application program interfaces and methods can be any suitable synchronization interface components designed for any particular application in PIM **7**. Because the application program interfaces are preferably standardized, they allow synchronization manager **110** to access and synchronize any number of different desktop PIMs, as long as the required interface methods are implemented for each PIM.

Reference store **112** provides a mapping between instances of objects stored in object store **6** on mobile device **3** and objects stored in object store **8** on desktop computer **4**. Since the same object identifiers are not used by PIM **5** to identify objects on object store **6** as are used by PIM **7** to identify objects in object store **8**, this mapping is required.

Synchronization manager **110** maintains reference store **112** so that reference store **112** contains the identifying data segments corresponding respectively to a plurality of object instances in object store **8** on desktop computer **4** that are to be synchronized with instances of the same object in object store **6** on mobile device **3**. The identifying data segments are updated each time corresponding object instances have been synchronized.

The exact composition of an identifying data segment which is used to identify the particular object instances are assignable by the developer of the desktop synchronization interface component **108**, and are then handled and stored by synchronization manager **110**. The identifying data segments preferably include some sort of time stamp information which can be compared to determine whether an object has changed since the identifying data segment was last recorded in reference store **112**.

In addition to maintaining a plurality of identifying data segments, synchronization manager **110** also maintains a list of object identifiers corresponding to objects maintained in object store **6**. These identifiers are provided to synchronization manager **110** whenever a new object is added to object store **6** on mobile device **3**.

SYNCHRONIZATION PROTOCOL

The exact protocol by which full synchronization is accomplished in accordance with one embodiment of the present system is described in greater detail in the above referenced patent application. However, a brief discussion of that protocol is helpful in understanding of the present invention. In order to synchronize objects, synchronization manager **110** first creates two lists of handles which refer to particular objects. The term "handle" refers to a number or other identifier that can be used to uniquely identify an object and to access the object. Generally, a handle is valid for a particular time period or session; such as during the time when an object has been "opened." If the same object is opened again, its handle may be different.

The first list of handles is obtained from reference store **112** and is indicative of objects which have been synchronized in the past and are identified in reference store **112**. The second list of handles is a list which identifies the objects stored on object store **8**. The two lists of handles are compared against one another to determine whether the same objects are stored in reference store **112** and object store **8**.

If an object is identified in reference store **112**, but not in object store **8**, that particular object has been deleted from the desktop **4** since the last synchronization. On the other hand, if an object is identified in object store **8**, but it does not appear in reference store **112**, then it has been added to the desktop since the last synchronization. In either case, synchronization manager **110** determines how to handle the object. In one preferred embodiment, those objects which have been deleted from desktop object store **8** are also deleted from reference store **112**. Further, those which have been added to object store **8** are also added to reference store **112**.

Synchronization manager **110** then determines whether any of the objects stored in object store **8** have been modified at the desktop since the last synchronization. In other words, if handles corresponding to the same object appear in both object store **8** and reference store **112**, but they are not identical (such as the time stamp, a revision number, or another suitable identifying segment is not the same) that indicates that the object in object store **8** has been modified since the last synchronization.

Synchronization manager **110** then determines whether any objects stored in object store **6** on mobile device **3** have been added or modified since the last synchronization. To determine whether an object has been added to object store **6**, synchronization manager **110** compares the list of objects in reference store **112** (which reflects all objects at the last synchronization) with a list of objects on object store **6** maintained by synchronization manager **102**. To determine whether an existing object has been modified, synchronization manager **102** is configured to maintain a status bit associated with each object stored in object store **6**. The status bit reflects whether the particular object associated with that bit has been changed since the last synchronization. If so, synchronization manager **102** notifies synchronization manager **110** of that change if device **3** is then coupled to desktop computer **4**, or simply logs the status bit and sends it to synchronization manager **110** the next time device **3** is coupled to desktop computer **4**.

It should be noted that none of these procedures require either synchronization manager **110** or synchronization manager **102** to be aware of the particular nature or format of the identifying data segments or of the objects to which they correspond. Rather, interface components **100** and **108** are called upon for all actions that depend upon the actual content of the identifying data segments, and the content of the objects. It is up to the designer of those interfaces to define a format for the identifying data segments that allows the interfaces to perform their required functions.

Once the changes, additions and deletions are determined by synchronization manager **110**, the items are synchronized. In order to do this, synchronization manager **110** forms a list of objects which have been changed on either object store **8** or object store **6** and simply calls upon the respective synchronization interface components to update the outdated object. If the same object has been modified both on mobile device **3** and desktop computer **4**, a conflict arises. Synchronization manager **110** resolves the conflicts by either prompting the user, or referencing profile information entered by the user during a set-up procedure which dictates which device the user wants to take precedence over the other, and in which instances. The process of setting up profile information is described in greater detail in the above-identification patent applications.

Where an object has either been created at the desktop computer **4** or in the mobile device **3**, that object needs to be exchanged with the other device. In the instance where mobile device **3** needs to obtain a new object from desktop computer **4**, synchronization manager **110** calls an interface method known as "Set-Up" which specifies a handle for the object to be obtained from object store **8** in desktop computer **4** and transferred to object store **6** in mobile device **3**. Once the handle is obtained, the method known as "Get-Packet" is called repeatedly to retrieve a data stream which represents the object, and which is formatted by interface component **108**. Synchronization manager **110** simply treats the data as a data stream which is retrieved and sent over link **9** to synchronization manager **102**, and eventually to object store **6**. The appropriate synchronization interface component **100** parses the data stream in order to identify certain property values associated with properties corresponding to the object. Those properties are then stored in the object store **6**.

Finally, synchronization manager **110** updates the identifying data segments associated with either synchronized or exchanged objects and stores the updated data segments in reference store **112**.

It is worth noting that the architecture described herein provides synchronization of objects associated with electronic mail messages, meetings or appointments, and contacts information, as well as other objects maintained by PIMs **5** and **7**. Thus, using the above synchronization protocol, objects associated with these features are synchronized during the synchronization process. Synchronization manager **110** detects any new objects in object store **8** which represent new electronic mail messages, meetings or appointments or contact information and causes those objects to be transferred to object store **6** on mobile device **3** for attention by the user of mobile device **3**.

Further, if the user composes an electronic mail message or schedules a meeting (and therefore creates a meeting object and an electronic mail meeting request object) or enters contact information on mobile device **3** using an appropriate application in PIM **5**, the objects associated with those items are stored as new objects in object store **6**. During synchronization, those new objects are transferred to desktop computer **4** and object store **8**.

MEETING REQUESTS

FIG. **6** is an architectural block diagram illustrating one preferred embodiment of a mechanism by which meeting requests are generated. Mobile device **3** and desktop computer **4** are shown coupled to one another, as shown in FIG. S. However, FIG. **6** also illustrates that mobile device **3** and desktop computer **4** can be coupled to a wide area network **138** which can include simply another mobile device, another desktop S computer, LAN **51** (shown in FIG. **2**), WAN **52** (shown in FIG. **2**), or any other suitable network.

Similar items to those shown in FIG. **5** are similarly numbered. However, in FIG. **6**, PIMs **5** and **7** are shown in greater detail. FIG. **6** illustrates that, in one preferred embodiment, PIM **5** includes an electronic mail application **140**, a calendar and scheduling application **142**, and an abridged address book **144**. Also, while each of the applications may have designated application programming interfaces, only a single API component **146** is shown in FIG. **6**, for the sake of simplicity.

FIG. **6** further illustrates that object store **6** preferably interfaces with applications **140**, **142** and **144** through its own set of application programming interfaces **148**. Thus, applications **140**, **142** and **144** maintain object store **6** by calling methods exposed by application programming interfaces **148** associated with object store **6**. In addition, communication component **94** and sync component **99** communicate with application programs **140**, **142** and **144** by calling methods exposed by application programming interfaces **146**.

PIM **7** in desktop computer **4** has also been illustrated as a more detail block diagram. In one preferred embodiment, PIM **7** includes electronic mail application program **150**, scheduling and calendaring program **152** and full address book program **154**. While each of the application programs **150**, **152** and **154** may have an associated set of application programming interfaces, only a single application programming interface component **156** is illustrated in FIG. **6**, for the sake of simplicity.

FIG. **6** further illustrates that, in one preferred embodiment, object store **8** is associated with its own set of application programming interfaces **158** (which may also be the same as API **156**, preferably MAPI, or which may be separate therefrom). Thus, application programs **150**, **152** and **154** call methods exposed by application programming interfaces **158** to maintain, and interact with, object store **8**. In addition, communications component **115** calls methods exposed by application programming interfaces **156** to interact with application programs **150**, **152** and **154**.

FIG. **6** shows that desktop computer **4** preferably includes electronic mail transport **160**. Electronic mail transport **160** is preferably a commercially available electronic mail transport, such as a SMTP transport. However, it should be noted that desktop computer **4** can include any suitable transport, or combination of transports.

FIG. **7** is a flow diagram illustrating the creation of a meeting request on mobile device **3**. The following description will proceed with reference to FIGS. **6** and **7**.

In order to create a meeting request from mobile device **3**, the user first enters meeting request information through a suitable user interface. In one preferred embodiment, the user opens the scheduling application program **142** which causes a suitable user interface to be displayed on screen **84**. Using stylus **86**, and possibly keypad **82** (or any other suitable input mechanism), the user enters appropriate information to request a meeting. Such information will typically include the date and time of the meeting, the requested attendees (which may be entered by using a fully qualified electronic mail address, or by entering a familiar name, or selected from entries in a contacts database which have a non-empty electronic mail address field), the subject matter of the meeting, an indication of whether the meeting is recurring or a single event, possibly the location of the meeting, and so on. Scheduling application **142** creates an object representative of the meeting request.

This information is used to create a meeting object and enter that object in the store associated with the calendar on mobile device **3**. As a result of the creation of the meeting object, an electronic mail meeting request object is also created. It should also be noted that subsequent critical modifications to the meeting object will also cause updated electronic mail meeting request objects to be created and transmitted as well.

In a preferred embodiment, a selection has already been made which specifies a transport by which any electronic mail meeting request objects are to be sent. Alternatively, the transport to be used can be implicit in the full electronic mail address, itself. For example, it can be specified that any of transports **132–136** are to be used, when mobile device **3** is to transmit the meeting request itself. In addition, a selection has also preferably already been made to specify certain options which are to be used in the sync protocol discussed above. For example, if an inbox synchronization option is enabled, the electronic mail meeting request objects will be synchronized to the outbox of desktop computer **4**. Also, if a calendar synchronization option is enabled, the meeting object will be synchronized to the calendar of desktop computer **4**. Desktop computer **4** then preferably transmits the electronic mail meeting request object synchronized to its outbox.

The receipt of the meeting request information from the user, and the creation of a meeting object representative of the meeting and an electronic mail meeting request object are indicated by blocks **162** and **164** in FIG. **7**.

Next, the critical time stamp is added to the objects (as indicated by block **165**) and scheduling application **142** assigns a global object identification tag to the meeting object created. The global object identification tag is a tag,

such as a number, which uniquely identifies the object representative of the meeting request to mobile device **3**, to desktop computer **4**, and to all other devices which may encounter that object. The global identification tag is carried with the electronic mail meeting request object when it is transmitted, and remains with the object on other devices. This is indicated by block **166**.

Because the electronic mail meeting request object must be transmitted to another device, scheduling application **142** calls methods in API **146** which manipulate electronic mail program **140** to retrieve a fully qualified electronic mail address for all of the potential attendees who are to receive the meeting request. In one preferred embodiment, scheduling application **142** obtains the fully qualified address directly from an address book. In another embodiment, mobile device **3** includes abridged address book program **144**. In a preferred embodiment, abridged address book program **144** is implemented as a contacts feature provided by Microsoft Outlook **97**. The abridged address book contains proper names, familiar names, addresses, telephone numbers, and fully qualified electronic mail addresses for people that the user has chosen to add to the abridged address book. Those people will likely be people who receive a significant number of electronic mail messages from the user of mobile device **3**. Therefore, that information likely contains the fully qualified electronic mail addresses for the potential attendees of the meeting request. In addition, if the address is not fully qualified, the synchronization component **97** on desktop **4** attempts to resolve the name on the desktop prior to sending the electronic mail transmission.

Thus, either electronic mail application **140** or scheduling application **142** call methods in API **146** which cause abridged address book program **144** to retrieve the fully qualified electronic mail address for the familiar names entered by the user as potential attendees. This significantly alleviates memory overhead which would otherwise be required, for instance, if the user was required to download a full address book from desktop computer **4** in order to send an electronic mail message or a meeting request. This is indicated by block **168**. In another preferred embodiment, the functionality associated with block **168** is performed at block **162**, as soon as the user inputs the appropriate information.

Also, the transport option has preferably already been chosen which indicates the particular transport by which the electronic mail messages are to be transmitted. Those transports can be any of transports **132–136** on mobile device **3** or through synchronization with desktop computer **4**, using one of its transports. Based on the option which has been chosen, the electronic mail meeting request object is transmitted through the appropriate transport and, once transmitted, is removed from the outbox of mobile device **3**.

If the electronic mail meeting request object is to be sent directly from one of the transports on mobile device **3**, the appropriate transport formats the meeting request for transmission. In a preferred embodiment, the particular properties which scheduling application program **142** attributes to an electronic mail meeting request object representative of a meeting request are compatible with multiple other PIMs. Therefore, these properties may be a subset or a superset, of all of the properties recognized by the multiple PIMs such that the object can be entertained and appropriately handled by those PIMs. Appendix A, which is attached hereto and forms a part of this document, is a list of message properties by which the Microsoft Schedule+ brand scheduling program defines objects associated with meeting requests. These properties notably include revised recurring notification properties which are backward compatible with other versions of Microsoft Schedule+ scheduling software. For instance, in the embodiment described in Appendix A, recurring notifications which are used to define recurring meetings are formatted as a superset of single-instance notifications. In this way, prior versions of the Microsoft Schedule+ scheduling software will be provided with well formed meeting notifications, even for recurring meetings.

In the preferred embodiment, scheduling program **142** then calls methods exposed by API **146** related to electronic mail application program **140**. The electronic mail message which contains the electronic mail meeting request object is formulated into one of a predetermined number of classes by electronic mail application **140**. The receiving scheduler program of the potential attendee differentiates between the type of meeting notification based on the mail message class. For example, in one preferred embodiment, mail message classifications exist for a meeting cancellation notice from the originator, a meeting request from the originator, a meeting acceptance from the attendee, a meeting declined from the attendee and a meeting tentatively accepted from the attendee. Thus, this is preferably included along with the object when the transport formats the electronic mail meeting request object for transmission. The meeting request is transmitted, through communications component **94** to the designated transport **132**, **134** or **136** which, in turn, formats and transmits the message to network **138**.

The formatting and transmission of the electronic mail meeting request object from a transport on mobile device **3** is indicated by blocks **172** and **174** in FIG. **7**.

After the electronic mail meeting request object is sent, it is removed from the outbox of mobile device **3** to indicate that the message has been sent. This is indicated by block **190**. Upon next being coupled to desktop computer **4**, the meeting object is synchronized to the store containing the calendar on desktop computer **4**. Preferably, the PIMs also provide a feature which can be utilized to prevent the PIM from creating and sending an electronic mail meeting request object simply because the meeting object has been synchronized to the calendar on the desktop. The feature may simply be to only create electronic mail meeting request objects for meeting requests created on the desktop. This feature is implemented to further address the problem of duplicate messaging.

If the transport option has been chosen to send the electronic mail meeting request object using the synchronization protocol and a mail transport on desktop computer **4**, processing is substantially the same, except that the electronic mail meeting request object is synchronized to the outbox of desktop computer **4** (as indicated by block **173**). Mobile device **3** simply waits until it is coupled to desktop computer **4** before any further action is taken with respect to that meeting request. Upon coupling of mobile device **3** to desktop computer **4**, the synchronization protocol described above is executed. During the synchronization process, the meeting object is synchronized to the calendar on the desktop computer **4** and the electronic mail meeting request object is then synchronized to the outbox of desktop computer **4** through synchronization components **97** and **99**. Upon receiving the electronic mail meeting request object indicative of the meeting request, electronic mail application program **150** recognizes that the meeting request needs to be transmitted. Electronic mail application program **150** calls the necessary methods exposed by API **158** to have the appropriate object transported, through communications component **115** and transport **160**, to network **138**. This is again indicated by blocks **172**, **173**, **174** and **190**.

It is also worth noting that, in one alternatively preferred embodiment, the meeting request need not be entirely formatted on mobile device **3**. Instead, the data representing the electronic mail meeting request object can simply be stored and transferred during synchronization to desktop computer **4** where it is fully formatted. For instance, in one preferred embodiment, mobile device **3** does not store the particular transport to be used by desktop computer **4** in sending the meeting request. Since this information is not stored by mobile device **3**, it is not synchronized with desktop computer **4**. Instead, electronic mail application program **150** on desktop computer **4** (prior to sending the meeting request) calls the necessary methods through API **156** to have full address book **154** retrieve the transport associated with each of the potential attendees identified by the user. That information is then used by electronic mail application program **150** in choosing the appropriate transport **160** with which to transmit the electronic mail meeting request object. In this way, additional memory capacity need not be consumed on mobile device **3** to store such transport information. This is indicated by block **188**.

In any case, once electronic mail application program **150** causes the electronic mail meeting request object to be transmitted to network **138**, it is removed from the outbox. This is indicated by block **190**.

After the electronic mail meeting request object has been sent it is determined whether any other meeting requests were synchronized with desktop computer **4** during the last synchronization process. If not, normal processing continues. If so, processing reverts back to block **172** wherein electronic mail application program **150** calls the necessary APIs to send the object. This is indicated by block **192**.

FIG. **8** is a flow diagram illustrating one preferred embodiment of how mobile device **3** handles responses to the meeting requests transmitted in accordance with FIG. **7**. In the preferred embodiment, the addressee for responses to meeting requests is the meeting originator. Both senders and recipients of the meeting requests can be arranged properly for delegate handling. The publicly available MAPI specification fully discloses how to interpret such properties.

Thus, the recipient of the meeting request is provided with a suitable user interface to indicate a response. The response is addressed to the meeting originator, or another proper delegate. The response is then transferred by the device which the recipient is operating on and is received through one of transports **132**, **134** or **136**, or through communications component **94** on mobile device **3**. This is indicated by block **194** in FIG. **8**.

Upon receiving the response, scheduling application program **142** correlates the response to the meeting object which was formed by scheduling application program **142** when the user created the request, and which is stored in object store **6**. This correlation is performed by using the unique global identification number which uniquely identifies that meeting request to all devices which may encounter it. This is indicated by block **196**.

The critical time stamp information is then checked to determine whether the response corresponds to the latest meeting object. The response preferably actually contains two time stamps. The first is the recipient's opinion of the originator's critical time stamp. The second is a time stamp assigned by the recipient. The first is checked to determined whether it is the same as the originator's time stamp. If so, the response is considered to be in date. If not, the response is rejected or ignored as an out-of-date response. The second must be equal to or greater than any previously recorded time stamp from this particular recipient for the response to be in date. If not, the response is rejected or ignored as out-of-date. This is indicated by block **197**.

It should be noted that in the preferred embodiment, the time stamps are always moving forward. Thus, even if the user resets the clock sufficiently into the past, any subsequent time stamp is always the later of a current clock time and the last time stamp plus one second.

Scheduling application program **142** then calls the necessary methods exposed by API **148** in order to update the attendee status associated with the request. This is updated based on the particular response received from the attendee. This is indicated by block **198**.

Mobile device **3** then simply waits to be connected to desktop computer **4**. Once it is connected, the updated objects (containing the updated attendee status property) are synchronized to the instance of that object stored in object store **8**. This is accomplished utilizing the synchronization protocol discussed above. This is indicated by blocks **200** and **202**. The meeting object is then available to scheduling application program **152** and desktop computer **4** to be displayed on the user's calendar at desktop computer **4** for user observation and interaction.

FIG. **9** is a flow diagram which illustrates one preferred embodiment of how responses are handled when received through transport **160** on desktop computer **4**. The user first inputs information through an appropriate user interface which defines the user's response to the meeting request. The response is then transmitted back to desktop computer **4** through transport **160** and communications component **115**. This is indicated by block **204**.

Scheduling application program **152** correlates the response to the meeting object, as did mobile device **3**, by utilizing the global identification number assigned to the meeting object, and received along with the response to the meeting request. This is indicated by block **206**. The time stamp information is then checked to determine whether the response corresponds to the latest meeting object, again as described above. If not, the response is rejected or ignored as an out-of-date response. This is indicated by block **207**. Scheduling application program **152** then calls the necessary methods exposed by API **158** to modify and update the attendee status of the meeting object stored in object store **8**. This is indicated by block **208**. Desktop computer **4** then simply waits until it is next coupled to mobile device **3**.

At that point, the updated objects in object store **8** (which have been updated to accurately reflect the new attendee status based on the response) are synchronized to the other instance of that object then stored in object store **6**. Synchronization is accomplished in accordance with the synchronization protocol discussed above. The updated object is then available for review and manipulation by the user of mobile device **3**. This is indicated by blocks **210** and **212**.

The mobile device in accordance with an illustrative embodiment of the present invention also handles exceptions to recurring meetings and meeting cancellations. In one embodiment, the scheduling PIM erases all exceptions when an update to the recurrence pattern of a recurrent meeting is received. Thus, new electronic mail objects must be created and sent for each exception once a change in the recurrent meeting pattern has been sent. For instance, assuming the user entered a recurring meeting request on mobile device **3**. Scheduling application **142** creates a recurring meeting object and enters it on the calendar of the user. Electronic mail application **140** also creates an electronic mail recurring meeting request object for transmission to

potential attendees. The recipients (potential attendees) can respond generally as described above.

For the sake of the present example, assume the recurring meeting is to occur every Tuesday from 10:00–11:00 a.m. beginning on April 1. However, the user (originator) then needs to cancel the April 8^{th} meeting. The user enters scheduling application **142** and deletes the desired instance. Electronic mail application **140**, in turn, creates another object for transmission which indicates this cancellation (which is an exception to the recurrence pattern). The originator then needs to move the April 15th meeting to 11:00–12:00 noon. A similar process is executed to create this exception as well. Suppose then that the originator wishes to make a change to the location of the recurring meetings and apply the change to the entire recurrence pattern. This requires three new pieces of electronic mail to be transmitted.

The first is simply created to indicate the new location which modifies the recurrence pattern. This may also, in some PIMs, erase the previous two exceptions. Thus, two new pieces of electronic mail are created to reinstitute the two exceptions (i.e., the cancellation of the April 8^{th} meeting and the time change for the April 15^{th} meeting). These pieces of electronic mail are automatically created by electronic mail application **140**. These same actions are taken in response to other suitable modifications as well, such as modification of the list of attendees.

LOCALIZATION

As previously discussed, in the preferred embodiment, a meeting object is typically accompanied by a textual phrase which describes the nature of the meeting request. The textual phrase is generated based upon the property values input by the user during creation of the meeting request. For example, a user may request the potential attendees to attend a meeting on the third Monday of each month at 10:00 a.m., and for a period which extends from Jan. 1, 2000, to Mar. 20, 2001. In that case, a typical text message which might be retrieved and placed in the meeting request object to accompany that object for display to the potential attendees might read as follows:

"This meeting shall take place on the third Monday of each month at 10:00 a.m. beginning Jan. 1, 2000, and ending Mar. 20, 2001."

However, in a different locality, convention may dictate that the day be placed before the month when reciting dates. Further, words or phrases in the text message, and the order of the components of the text message, may need to be rearranged in order to more closely conform to local convention.

In the preferred embodiment, a predetermined text message of the type set out above is generated to describe each different type of recurring meeting which can be requested by the user of mobile device **3**.

In accordance with one aspect of the present invention, the text messages used to describe requests for recurring meetings are broken down into a plurality of phrases. Each of the phrases which has been created is broken down into a plurality of fields. The particular phrases which can be created to define a meeting request, and the particular fields into which those phrases are separated can be any suitable phrases and fields.

For the phrase mentioned above, FIG. **11**A illustrates one embodiment of the fields into which that phrase can be broken. FIG. **11**A shows that a first portion of the phrase "This meeting shall take place on the third Monday of each month" is placed in field **214**. The term "at" is placed in field **216**. The time designation "10:00" is placed in field **218** and the AM/PM designator is placed in field **220**. The term "beginning" is placed in field **222**, and the remaining message is similarly broken up into fields **224**, **226**, **228**, **230**, **234**, **236**, **238**, **240** and **242**.

FIG. **10** is a flow diagram illustrating how the localization process works. The creation of the textual phrases defining meeting requests, and the breaking up of those requests into various predetermined fields, is indicated by block **242**. Next, during manufacture of mobile device **3**, a localizer program is run on mobile device **3** to rearrange fields **214–240**, for each of the messages created, such that the messages conform to local convention. Of course, the specific localizer program which is run on mobile device **3** will depend on the particular locality and consumer base for which mobile device **3** is intended. Once the localizer program has been run, mobile device **3** contains a plurality of sets of fields, referred to as templates, into which message data is placed in order to display the textual message in a localized fashion. Running of the localizer program in setting up the templates is indicated by block **244**.

After mobile device **3** has templates which have been localized, mobile device **3** is ready to receive a meeting request and properly display the textual description of the meeting request. When a meeting request is received, the data stream indicative of the textual phrase which describes the meeting is also received by scheduling application program **142**. This is indicated by block **246**. Scheduling application program **142** parses the incoming data stream into values associated with the various fields in the appropriate template (the appropriate template corresponds to the specific textual description being received) This is indicated by block **248**.

Next, the parsed data is placed into preselected fields in the templates based on the created phrases. This is indicated by block **250**. Because the localizer program has already been run, the fields in the templates are already placed in appropriate order to conform to local convention.

FIG. **11**B illustrates the rearrangement of the fields in the template illustrated in FIG. **11**A to conform to a different convention. FIG. **11**B illustrates the conformation of the phrase to a convention in which the phrasing is slightly different from that shown in FIG. **11**A, and in which the day is conventionally placed prior to the month in a date. The phrase now contains the same terms, but reads:

"Beginning Jan. 1, 2000 and ending Mar. 20, 2001 this meeting shall take place on the third Monday of each month at 10:00 AM."

All appropriate phrases will, of course, be similarly localized.

Also, in an illustrative embodiment, time zone information is handled in an advantageous manner. For instance, when a recurring meeting request is created and sent to a device located in a different time zone, the textual phrase indicative of the recurring meeting includes an indication of the time zone in which the recurring meeting request was created. For instance, if a user in Paris sends a recurring meeting request to a recipient in Seattle (nine hours different) for a meeting occurring every Tuesday at 8:00 a.m. beginning March 3, the textual phrase may be:

"This meeting occurs every Tuesday at 8:00 a.m. beginning March 3; (GMT+1) Paris, Madrid."

This allows the recipient to view the recurrence pattern as originally composed by the sender.

On the other hand, if the meeting is non-recurrent, the time is preferably simply displayed as the appropriate time

for the recipient (e.g., "This meeting is scheduled for Tuesday at 11:00 p.m.").

CONCLUSION

Therefore, it can be seen that the present invention provides significant advantages over prior systems. First, the present invention allows the creation of a meeting request from a mobile device itself. The present invention further identifies the object associated with such a meeting request in a manner which uniquely identifies the object to all other devices likely to encounter the object. The present invention also utilizes a feature in the PIMs which indicates whether the meeting request has been sent to potential attendees. These features eliminate duplicate messaging.

The present invention also preferably implements features which reduce the memory required to implement the meeting request feature by use of an abridged address book. Further, the present invention preferably uses a set of scheduling properties which are compatible with a plurality of different PIMs likely to encounter the meeting request. In addition, the present invention preferably utilizes localization procedures which localize meeting requests to more closely conform to local convention and handle time zone information in an advantageous manner. The present invention also preferably supports the use of multiple electronic mail transports as well as a synchronization protocol.

Although the present invention has been described with reference to preferred embodiments, workers skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.

What is claimed is:

1. A mobile device, comprising:

an object store;

an application program configured to maintain objects on the object store;

a user input mechanism configured to receive user input information;

a synchronization component configured to synchronize individual objects stored on the object store with remote objects stored on a remote object store;

a communications component configured to communicate with a remote device containing the remote object store; and

wherein the application program is further configured to generate a meeting object and an electronic mail scheduling request object based on the user input information.

2. The mobile device of claim **1** wherein the application program is configured to generate the meeting object with a global identifier property uniquely identifying the meeting object among a plurality of other objects.

3. The mobile device of claim **2** wherein the application program is configured to generate the meeting object with a time stamp indicative of a relative time when the meeting object was created and wherein the application program generates the electronic mail scheduling request object with the time stamp.

4. The mobile device of claim **3** wherein the application program comprises:

a first application program configured to generate the meeting object based on the user input information; and

a second application program configured to generate the electronic mail meeting request object.

5. The mobile device of claim **1** wherein the application program further comprises:

a contacts application program configured to maintain objects on the object store indicative of contact information wherein the contact information includes address information indicative of a fully qualified electronic mail addresses for individuals identified by the contact information; and

wherein the application program is configured to obtain the fully qualified electronic mail address of potential attendees identified by the contact information by interaction with the contacts application program.

6. The mobile device of claim **1** wherein the application program is configured to generate the meeting object and the electronic mail scheduling request object such that properties of the objects are compatible with at least a second application program associated with the remote object store and different from the application program.

7. The mobile device of claim **1** wherein the application program is configured to receive a data stream indicative of a textual phrase describing the meeting object, to parse the data stream into sections and place the sections in corresponding fields of a preselected template containing the fields, the preselected template being associated with the textual phrase received.

8. The mobile device of claim **7** wherein the preselected template is created by arranging the fields in an order, the order being based on a specific locality.

9. A method of operating a mobile device, comprising:

providing a first object store on the mobile device;

providing a first application program on the mobile device;

maintaining objects in the first object store with the first application program;

intermittently synchronizing the objects in the first object store with objects in a remote object store;

receiving user input information indicative of a meeting request;

generating a meeting object with the first application program such that at least some of the user input information defines properties in the meeting object;

generating an electronic mail meeting request object based on the information in the meeting object; and

storing the meeting object and the electronic mail scheduling request object in the first object store for transmission.

10. The method of claim **9** wherein synchronizing comprises:

coupling the mobile device to a computing device having the remote object store;

synchronizing objects in the first data store with objects in the remote data store; and

transmitting the electronic mail scheduling request object from an electronic mail transport on the computing device.

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

providing an electronic mail transport on the mobile device; and

transmitting the electronic mail scheduling request object through an electronic mail transport on the mobile device.

12. The method of claim **11** wherein providing an electronic mail transport on the mobile device comprises:

providing a plurality of electronic mail transports on the mobile device; and

selecting one of the plurality of electronic mail transports through which the electronic mail scheduling request objects are to be transmitted.

13. The method of claim 9 wherein generating a meeting object comprises:

assigning the meeting object a global identifier which uniquely identifies the meeting request relative to other objects.

14. The method of claim 13 wherein generating a meeting object further comprises:

assigning the meeting object a time stamp indication indicating a time when the scheduling object was created; and

wherein generating the electronic mail meeting request object includes assigning the electronic mail meeting request object the time stamp indication.

15. The method of claim 14 and further comprising:

receiving response objects;

correlating the response objects with the meeting object on the mobile device based on the global identifier and the time stamp indication; and

updating a response status associated with the meeting object based on the response objects received.

16. The method of claim 15 and further comprising: synchronizing the response status with the remote object store.

17. A data transmission system, comprising:

a first computing device including:

a first data store configured to store objects;

a user input mechanism; and

a first application program configured to receive user input information from the user input mechanism, create a first object based on the user input information and store the first object on the first data store;

a synchronization manager configured to synchronize objects in the first data store with objects in a second data store;

a second computing device including: the second data store, the second data store being configured to store objects; and

a second application program configured to access the second data store and create an electronic mail scheduling object based on the first object being synchronized to the second data store from the first data store;

an electronic mail transport; and

wherein the second application program is configured to transmit the electronic mail scheduling objects with the electronic mail transport.

18. The system of claim 17 and further comprising:

a third computing device including:

a third data store configured to store objects; and

a third application program configured to access the third data store, to receive electronic mail scheduling objects from the second computing device and to store the electronic mail scheduling objects on the third data store.

19. The system of claim 18 and further comprising:

a fourth computing device including:

a fourth data store; and

a fourth application program configured to access the fourth data store and store objects on the fourth data store; and

wherein the synchronization manager is configured to synchronize objects in the third and fourth data stores.

20. The system of claim 19 wherein the synchronization manager comprises:

a first synchronization manager on at least one of the first and second computing devices; and

a second synchronization manager on at least one of the third and fourth computing devices.

21. The system of claim 17 wherein the first object comprises a meeting object and wherein the electronic mail scheduling object comprises an electronic mail meeting request object.

22. The system of claim 17 wherein the first computing device comprises a mobile device.

23. The system of claim 22 wherein the first application program comprises a scheduling program and wherein the second application program comprises an electronic mail application program.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,370,566 B2
DATED : April 9, 2002
INVENTOR(S) : Discolo et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page,
Item [75], Inventors, "Bellevue" should be -- Clinton --

Column 10,
Line 57, "30" should be -- 3 --

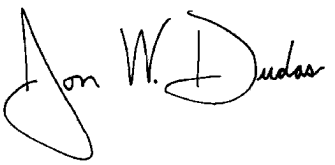
Column 11,
Line 13, "Vise" should be -- vice --

Column 15,
Line 31, "S" should be -- 5 --
Line 34, delete "S"

Column 22,
Line 46, "Jan. 1" and "Mar. 20" should be -- 1 Jan. -- and -- 20 Mar. --

Signed and Sealed this

Twenty-fifth Day of January, 2005



JON W. DUDAS
Director of the United States Patent and Trademark Office

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

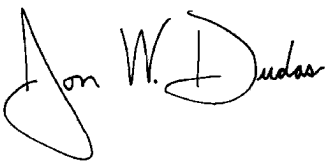
PATENT NO. : 6,370,566 B2
APPLICATION NO. : 09/058679
DATED : April 9, 2002
INVENTOR(S) : Discolo et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 23,
Line 32, insert Appendix A as shown on attached pages.

Signed and Sealed this

Twentieth Day of June, 2006



JON W. DUDAS
Director of the United States Patent and Trademark Office

APPENDIX A

{autonumlgl } Audience/About this document

This document describes the message properties used by Schedule+ 2.0 to transmit meeting information to another Schedule+ 2.0 user. The document also describes the Schedule+ 1.0 counterpart message properties (where appropriate).

The reader should be familiar with MAPI, specifically properties stored on messages. The reader should also be somewhat familiar with MAPI 0 properties and how Schedule+ 1.0 used them to transmit meeting information.

{autonumlgl } Consumers

This document is intended for third-party developers who plan to interact with Schedule+ 2.0 via email.

{autonumlgl } The Design

Schedule+ 2.0 stores properties of a meeting in similarly typed MAPI properties in a message. Typically, the subject of the message matches the description of the meeting (as it did 1.0), and the body of the message contains the meeting notes (new for 2.0). The addressees of a meeting request or cancellation are compiled from the list of attendees in the meeting, filtered on who should actually receive this mail. The addressee for meeting responses is the meeting originator. Senders and recipients are arranged properly for delegate handling. See the MAPI spec on how to interpret these standard properties.

For mail systems that do not automatically handle delegation of mail, Schedule+ 2.0 will open each addressee's schedule file (or equivalent) to determine if mail should be sent to a delegate instead of or in addition to the intended recipient. Also, Schedule+ 2.0 will determine if the sending user is mailing on behalf of someone else and stamp appropriate properties into the message. The MAPI 1.0 properties which control delegation are PR_SENT_REPRESENTING_(ENTRYID/NAME) and PR_RCVD_REPRESENTING_(ENTRYID/NAME). See the MAPI 1.0 spec on how these properties are set to induce delegation.

Beyond the standard properties, Schedule+ 2.0 defines and uses private properties which have no "normal" messaging counterpart. There are two basic kinds of communication between a meeting organizer and an attendee: some notification related to a single meeting, or some notification related to a recurring meeting. For backward compatability, the recurring notifications are formatted as a superset of single-instance notifications. This way, Schedule+ 1.0 users will at least see a well-formed meeting notification.

As in version 1.0, Schedule+ 2.0 differentiates the type of meeting notification by the mail message class. The class names have changed slightly since 1.0.

Version 1.0 Class name	Version 2.0 class name	Description
IPM.Microsoft Schedule.MtgCncl	IPM.Schedule.Meeting.Canceled	Meeting Cancellation notice. From originator
IPM.Microsoft Schedule.MtgReq	IPM.Schedule.Meeting.Request	Meeting Request. From originator

{autonumlgl } PR_WHERE

A descriptive string describing where the meeting will be held (no 1.0 counterpart). The text of this string will also appear in the body of the note so that non-Schedule+ users will still see some useful information. In 2.0, the embedded body text will be removed before it is displayed or written into the schedule.

{autonumlgl } PR_GLOBAL_OBJID

a "global" ID for the meeting. This ID uniquely and universally ID's the meeting or recurrence pattern. Its internal structure is:

+0 bytes: GUID identifying client vendor. For Microsoft Schedule+ 2.0, this GUID is

0x04, 0x82, 0xE0, 0x74, 0xC5, 0xB7, 0x10, 0x1A,
0x82, 0xE0, 0x08, 0x00, 0x2B, 0x36, 0xA3, 0x33

Other vendors should use their own GUID.

+16 bytes: Implementation specific data.

If this property is missing, the attendee tracking features will be disabled. This means further notifications concerning this meeting will not be properly handled by the recipient, and the recipient will not be able to communicate attendee status properly with the originator.

{autonumlgl } PR_REQUIRED_ATTENDEES

If present, lists all the invited "required" attendees. The string is a concatenated form of the display names of each required attendee, separated by semicolons. The intent is the string itself may be placed in the "To:" well of a mail message when the original meeting request is Replied or Reply-All'ed, and the names will directly resolve into real recipients.

{autonumlgl } PR_OPTIONAL_ATTENDEES

If present, lists all the invited "optional" attendees. The string is a concatenated form of the display names of each optional attendee, separated by semicolons. The intent is the string itself may be placed in the "CC:" well of a mail message when the original meeting request is Replied or Reply-All'ed, and the names will directly resolve into real recipients.

{autonumlgl } PR_RESOURCE_ATTENDEES

If present, lists all the invited "resource" attendees. The string is a concatenated form of the display names of each resource attendee, separated by semicolons. The intent is the string itself may be placed in the "BCC:" well of a mail message when the original meeting request is Replied or Reply-All'ed, and the names will directly resolve into real recipients.

{autonumlgl } PR_PROCESSED

If present and TRUE, indicates that the message's contents have been processed (ie, for responses, the user's response code has been incorporated back into the owner's schedule). This property is not transmitted and is written only by the Schedule+ 2.0 client after the message has successfully been processed.

IPM.Microsoft Schedule.MtgRespP	IPM.Schedule.Meeting.Resp.Pos	Meeting Acceptance. From attendee.
IPM.Microsoft Schedule.MtgRespN	IPM.Schedule.Meeting.Resp.Neg	Meeting Declined. From attendee
IPM.Microsoft Schedule.MtgRespA	IPM.Schedule.Meeting.Resp.Tent	Meeting Tentatively Accepted. From attendee

{autonumlgl}Properties on Single Instance Meetings

The following properties appear on all Schedule+ 2.0 messages and describe a single instance of a meeting. For a recurring meeting requests, the single instance data will be the first instance of the recurrence. For recurrence exceptions, the data will be for that exception.

Some of the following data is optional only in that the notification remains useful even if the data is missing. Data that must be present and valid is marked as so.

{autonumlgl} PR_START_DATE

UTC-adjusted starting date/time of the appointment. Must be present. Uses Schedule+'s method of converting the time from the local timezone into UTC (the NT algorithm is buggy in that it uses the current system time to determine whether or not to consider Daylight adjustments).

{autonumlgl} PR_END_DATE

UTC-adjusted ending date/time of the appointment. Must be present.

{autonumlgl} PR_OWNER_CRITICAL_CHANGE

This is an increasing value based on the time the appoinment date, time, or where data was last modified on the owner's schedule. It is used to insure responses and requests are processed in the correct order (mail receive or send order does not work properly). It is only ever changed by the owner of a meeting. It must be present on all meeting mail.

{autonumlgl} PR_ATTENDEE_CRITICAL_CHANGE

This is an increasing value based on the time of the last response sent by the attendee. It is used to insure responses are processed in the correct order (mail receive or send order does not work properly). It is only ever changed by the invited attendee of a meeting. It must be present on all Meeting Responses.

The idea behind the last two properties is that when a meeting request is sent, it contains in PR_OWNER_CRITICAL_CHANGE the last modification time of the appointment according to the owner's clock. When a recipient reads the reequest, the value in PR_OWNER_CRITICAL_CHANGE is compared against the previous value (from a prior booking) to see if the request is still in date. If it is, the recipient may respond as desired. The response will have a copy of the same value in PR_OWNER_CRITICAL_CHANGE and will stamp the current time in PR_ATTENDEE_CRITICAL_CHANGE (according to the recipient's clock). When the owner reads the response, the PR_OWNER_CRITICAL_CHANGE will be compared against the timestamp stored in the appointment to decide if the response corresponds with this version of the appointment. Then, PR_ATTENDEE_CRITICAL_CHANGE is compared against any prior timestamp for responses received from this user to decide if the response itself is current.

{autonumlgl } PR_IS_SILENT

If present and TRUE, indicates that the message's contents are intended solely for the Schedule+ 2.0 client. There is no message content of interest for the user. This typically applies to response mail in which the attendee didn't type anything into the message body for the meeting owner to read. Therefore, the mail should be processed and deleted automatically.

{autonumlgl } PR_WANT_SILENT_RESP

This is used in tandem with PR_RESPONSE_REQUESTED. It is expected that messages from Schedule+ 1.0 will not have this property. Unless this property is present and TRUE, responses will only be sent if PR_RESPONSE_REQUESTED is also present and TRUE. Also, those responses will necessarily be non-silent (ie, the PR_IS_SILENT property in the response will be either missing or FALSE).

{autonumlgl } PR_DELEGATE_MAIL

If present and TRUE, indicates that the message was sent as part of Schedule+ 2.0's sender-side delegation strategy. If someone receives a piece of mail without this property marked true, but does have someone listed as being a delegate, the mail will automatically be forwarded to that delegate. This scenario can happen if someone sends a piece of mail, and the recipient adds someone as a delegate before the mail is received (but after it has been sent).

{autonumlgl }Properties used with Schedule+ 1.0

The properties listed here need to be translated on the fly to/from the old 1.0 format when the message is converted to/from Mail 3.0 form. This happens by the TNEF converter in MAPI, and shouldn't be the concern of any other developers.

{autonumlgl } PR_RESPONSE_REQUESTED

If present and TRUE, indicates that the meeting owner has requested that attendees send a response with meaningful text (ie, PR_IS_SILENT is expected to be false). If the property is missing or FALSE, it indicates the sender (owner) is not interested in any response details beyond the actual response code. The recipient (attendee) will not be given the option to write up response mail. However, a response message will be sent and auto-processed by the owner (PR_IS_SILENT will be true). The recipient will be able to override this behavior with a menu option.

{autonumlgl } PR_OWNER_APPT_ID

A Schedule+ 1.0-spefic form of an appointment ID. Schedule+ 2.0 will write a semi-valid ID here (only in that it probably won't cause problems when communicating with 1.0). Other clients should leave this property blank.

{autonumlgl }Properties on Recurring Meetings

{autonumlgl } PR_IS_RECURRING

If present and TRUE, indicates that this meeting request contains all the following data.

{autonumlgl } PR_IS_EXCEPTION

If present and TRUE, indicates that this request is for a single instance of a pattern.

{autonumlgl } PR_SINGLE_INVITE

If present and TRUE, indicates that this request is an invitation to the single instance only. The recipient is not invited to the rest of the pattern.

{autonumlgl } PR_TIME_ZONE

Desribes the owner's time zone (TBD - no work on time zones has been done yet). Must be present for recurring meeting requests.

{autonumlgl } PR_START_RECUR_DATE

The start date of the recurrence pattern in the originator's local time zone. It's encoded by: ((long)ymd.yr * 16l + (long)ymd.mon) * 32l + (long)ymd.day. It's stored as a long so gateways won't interpret the date as if it were in GMT and possibly muck with it. Must be present for recurring meeting requests.

{autonumlgl } PR_START_RECUR_TIME

The start time of the pattern in the originator's local time zone encoded by: ((long)time.hr * 64l + (long)time.min) * 64l + (long)time.sec. Same reason for encoding. Must be present for recurring meeting requests.

{autonumlgl } PR_END_RECUR_DATE

If present, holds the end date for the pattern. If missing, the pattern runs forever. See PR_START_RECUR_DATE for encoding method.

{autonumlgl } PR_END_RECUR_TIME

The end time of the pattern. See PR_START_RECUR_TIME for encoding. Note that unless the first instance of the pattern is an exception, the start date/time and end time for the pattern will coincide with PR_START_DATE and PR_END_DATE in this message. Must be present for recurring meeting requests.

{autonumlgl } PR_DOW_PREF

Describes the 1st day of week user preference. This is for multi-day weekly patterns that have an interval greater than one. Imagine a pattern that's every two weeks on Sunday and Monday starting on the 1st. If the first day of week is Sunday the 1st, the pattern will consist of 1, 2, 15, 16, etc. If the first day of week is Monday, the pattern will be 2, 8, 16, 22, etc. Must be present for recurring meeting requests which describe a weekly pattern where the week interval is greater than 1.

{autonumlgl } PR_RECUR_TYPE

Describes the type of recurrence. One of trecurDaily (0x40), trecurWeekly (0x30), trecurMonthly1 (0x0C), trecurMonthly2 (0x38), trecurYearly1 (0x07), trecurYearly2 (0x33). Must be present for recurring meeting requests.

{autonumlgl } PR_DAY_INTERVAL

Describes the delta in days between instances of this pattern. If missing and required for the pattern type, it is assumed to be 1.

{autonumlgl } PR_WEEK_INTERVAL

Describes the delta in weeks between instances of this pattern. If missing and required for the pattern type, it is assumed to be 1.

{autonumlgl } PR_MONTH_INTERVAL

Describes the delta in months between instances of this pattern. If missing and required for the pattern type, it is assumed to be 1.

Appendix B: Mapping from Schedule+ 1.0 properties

The following table associates the Schedule+ 1.0 message properties to their 2.0 counterparts. All of these properties are defined by MAPI 1.0 for backwards compatibility. Please see the MAPI 1.0 documentation for a description of the PR_SENT_REPRESENTING_* and PR_RCVD_REPRESENTING_* properties.

Schedule+ 1.0	Schedule+ 2.0
attOwner	if msg is a mtg req or cancellation: PR_SENT_REPRESENTING_(ENTRYID/NAME) otherwise: PR_RCVD_REPRESENTING_(ENTRYID/NAME)
attSentFor	only if msg is not a mtg req or cancellation PR_SENT_REPRESENTING_(ENTRYID/NAME)
attDelegate	no counterpart
attWhen	no counterpart (string recomputed from other data)
attAidLocal	no counterpart
attDateStart	PR_START_DATE
attDateEnd	PR_END_DATE
attAidOwner	PR_OWNER_APPT_ID
attRequestRes	PR_RESPONSE_REQUESTED

Appendix C: Schedule+ 1.0 property definitions

```
#define attOwner        FormAtt ( iattClientMin+0, atpByte )
#define attSentFor      FormAtt ( iattClientMin+1, atpByte )
#define attDelegate     FormAtt ( iattClientMin+2, atpByte )
#define attWhen         FormAtt ( iattClientMin+4, atpString )
#define attAidLocal     FormAtt ( iattClientMin+5, atpLong )
#define attDateStart    FormAtt ( iattClientMin+6, atpDate )
#define attDateEnd      FormAtt ( iattClientMin+7, atpDate )
#define attAidOwner     FormAtt ( iattClientMin+8, atpLong )
#define attRequestRes   FormAtt ( iattClientMin+9, atpShort )
```

See the MAPI 0 documentation for the meaning of these macros.

{autonumlgl } PR_YEAR_INTERVAL

Describes the delta in years between instances of this pattern. If missing and required for the pattern type, it is assumed to be 1.

{autonumlgl } PR_DOW_MASK

Bit field (bit 0 = Sunday) which describes which days or the week are valid in this pattern. If missing and required for the pattern type, it is assumed to be 2^8-1 (all days).

{autonumlgl } PR_DOM_MASK

Bit field (bit 0 = day 1) which describes which days of any given month are valid in this pattern. If missing and required for the pattern type, it is assumed to be 2^{32}-1 (all days).

{autonumlgl } PR_MOY_MASK

Bit field (bit 0 = January) which describes which months in any given year are valid in this pattern. If missing and required for the pattern type, it is assumed to be 2^{12}-1 (all months).

Appendix A: Property Definitions

Properties defined by MAPI for S+ backwards compatibility:

PR_START_DATE
PR_END_DATE
PR_RESPONSE_REQUESTED
PR_OWNER_APPT_ID

Non-transmitted properties used by S+:

#define PR_PROCESSED PROP_TAG(PT_BOOLEAN, 0x7d01)

We use named properties for the remaining definitions.

GUID = {0x6FD8DA90, 0x450B, 0x101B, 0x98, 0xDA, 0x00, 0xAA, 0x00, 0x3F, 0x13, 0x05}

Name	LID	PropType
ATTENDEE_CRITICAL_CHANGE	1	PT_SYSTIME
WHERE	2	PT_TSTRING
GLOBAL_OBJID	3	PT_BINARY
IS_SILENT	4	PT_BOOLEAN
IS_RECURRING	5	PT_BOOLEAN
REQUIRED_ATTENDEES	6	PT_TSTRING
OPTIONAL_ATTENDEES	7	PT_TSTRING
RESOURCE_ATTENDEES	8	PT_TSTRING
DELEGATE_MAIL	9	PT_BOOLEAN
IS_EXCEPTION	10	PT_BOOLEAN
SINGLE_INVITE	11	PT_BOOLEAN
TIME_ZONE	12	PT_I4
START_RECUR_DATE	13	PT_I4
START_RECUR_TIME	14	PT_I4
END_RECUR_DATE	15	PT_I4
END_RECUR_TIME	16	PT_I4
DAY_INTERVAL	17	PT_I2
WEEK_INTERVAL	18	PT_I2
MONTH_INTERVAL	19	PT_I2
YEAR_INTERVAL	20	PT_I2
DOW_MASK	21	PT_I4
DOM_MASK	22	PT_I4
MOY_MASK	23	PT_I4
RECUR_TYPE	24	PT_I2
DOW_PREF	25	PT_I2
OWNER_CRITICAL_CHANGE	26	PT_SYSTIME

```
/*
 *  S P L U S T A G S . H
 *
 *  Property tag definitions for standard properties of Schedule+ 2.0
 *  objects.
 *
 *  Copyright 1986-1997 Microsoft Corporation. All Rights Reserved.
 */

#ifndef SPLUSTAGS_H
#define SPLUSTAGS_H


#define HHPR_START_DATE                                 PROP_TAG( PT_SYST
IME,    0x0060)
#define HHPR_END_DATE                                   PROP_TAG( PT_SYST
IME,    0x0061)
#define HHPR_OWNER_APPT_ID                              PROP_TAG( PT_LONG
,       0x0062)
#define HHPR_RESPONSE_REQUESTED                         PROP_TAG( CEVT_I2
,       0x0063)

#define HHPR_OWNER_CRITICAL_CHANGE                      PROP_TAG( CEVT_FI
LETIME,  0x0064)
#define HHPR_ATTENDEE_CRITICAL_CHANGE                   PROP_TAG( CEVT_FI
LETIME,  0x0065)
#define HHPR_WHERE                                      PROP_TAG( CEVT_LP
WSTR,    0x0066)
#define HHPR_GLOBAL_OBJID                               PROP_TAG( CEVT_BL
OB,      0x0067)
#define HHPR_REQUIRED_ATTENDEES                         PROP_TAG( CEVT_LP
WSTR,    0x0068)
#define HHPR_OPTIONAL_ATTENDEES                         PROP_TAG( CEVT_LP
WSTR,    0x0069)
#define HHPR_RESOURCE_ATTENDEES                         PROP_TAG( CEVT_LP
WSTR,    0x006a)

#define HHPR_IS_SILENT                                  PROP_TAG( CEVT_I2
,        0x006c)
#define HHPR_WANT_SILENT_RESP                           PROP_TAG( CEVT_I2
,        0x006d)
#define HHPR_DELEGATE_MAIL                              PROP_TAG( CEVT_I2
,        0x006e)
#define HHPR_IS_RECURRING                               PROP_TAG( CEVT_I2
,        0x006f)
#define HHPR_IS_EXCEPTION                               PROP_TAG( CEVT_I2
,        0x0070)
#define HHPR_SINGLE_INVITE                              PROP_TAG( CEVT_I2
,        0x0071)
#define HHPR_TIME_ZONE                                  PROP_TAG( CEVT_I4
,        0x0072)
#define HHPR_START_RECUR_DATE                           PROP_TAG( CEVT_I4
```

```
,        0x0073)
#define HHPR_START_RECUR_TIME                          PROP_TAG( CEVT_I4
,        0x0074)
#define HHPR_END_RECUR_DATE                            PROP_TAG( CEVT_I4
,        0x0075)
#define HHPR_END_RECUR_TIME                            PROP_TAG( CEVT_I4
,        0x0076)
#define HHPR_DOW_PREF                                  PROP_TAG( CEVT_I2
,        0x0077)
#define HHPR_RECUR_TYPE                                PROP_TAG( CEVT_I2
,        0x0078)
#define HHPR_DAY_INTERVAL                              PROP_TAG( CEVT_I2
,        0x0079)
#define HHPR_WEEK_INTERVAL                             PROP_TAG( CEVT_I2
,        0x007a)
#define HHPR_MONTH_INTERVAL                            PROP_TAG( CEVT_I2
,        0x007b)
#define HHPR_YEAR_INTERVAL                             PROP_TAG( CEVT_I2
,        0x007c)
#define HHPR_DOW_MASK                                  PROP_TAG( CEVT_I4
,        0x007d)
#define HHPR_DOM_MASK                                  PROP_TAG( CEVT_I4
,        0x007e)
#define HHPR_MOY_MASK                                  PROP_TAG( CEVT_I4
,        0x007f)

// Non-transmitted property (not named)
#define HHPR_PROCESSED                                 PROP_TAG( CEVT_I2
,   0x7d01)

#ifndef HHPR_TAG_ONLY

static ULONG SPlusNamedTagTypes[30] = {
    PT_SYSTIME,
    PT_STRING8,
    PT_BINARY,
    PT_BOOLEAN,
    PT_BOOLEAN,
    PT_STRING8,
    PT_STRING8,
    PT_STRING8,
    PT_BOOLEAN,
    PT_BOOLEAN,
    PT_BOOLEAN,
    PT_I4,
    PT_I4,
    PT_I4,
    PT_I4,
    PT_I4,
    PT_I2,
    PT_I2,
    PT_I2,
```

```
    PT_I2,
    PT_I4,
    PT_I4,
    PT_I4,
    PT_I2,
    PT_I2,
    PT_SYSTIME,
    PT_SYSTIME,
    PT_SYSTIME,
    PT_BOOLEAN,
    PT_I4 };

static ULONG SPlusNamedTags[30] = {
    HHPR_ATTENDEE_CRITICAL_CHANGE,
    HHPR_WHERE,
    HHPR_GLOBAL_OBJID,
    HHPR_IS_SILENT,
    HHPR_IS_RECURRING,
    HHPR_REQUIRED_ATTENDEES,
    HHPR_OPTIONAL_ATTENDEES,
    HHPR_RESOURCE_ATTENDEES,
    HHPR_DELEGATE_MAIL,
    HHPR_IS_EXCEPTION,
    HHPR_SINGLE_INVITE,
    HHPR_TIME_ZONE,
    HHPR_START_RECUR_DATE,
    HHPR_START_RECUR_TIME,
    HHPR_END_RECUR_DATE,
    HHPR_END_RECUR_TIME,
    HHPR_DAY_INTERVAL,
    HHPR_WEEK_INTERVAL,
    HHPR_MONTH_INTERVAL,
    HHPR_YEAR_INTERVAL,
    HHPR_DOW_MASK,
    HHPR_DOM_MASK,
    HHPR_MOY_MASK,
    HHPR_RECUR_TYPE,
    HHPR_DOW_PREF,
    HHPR_OWNER_CRITICAL_CHANGE,
    HHPR_START_DATE,
    HHPR_END_DATE,
    HHPR_RESPONSE_REQUESTED,
    HHPR_OWNER_APPT_ID};

#endif  // HHPR_TAG_ONLY

#endif  /* SPLUSTAGS_H */
```