US 20020049760A1

(19) **United States**

(12) **Patent Application Publication**

Scott et al.

(10) **Pub. No.: US 2002/0049760 A1**

(43) **Pub. Date: Apr. 25, 2002**

(54) **TECHNIQUE FOR ACCESSING INFORMATION IN A PEER-TO-PEER NETWORK**

(75) Inventors: **Adrian C.H. Scott**, San Francisco, CA (US); **S. Mitra Ardron**, Fairfax, CA (US)

Correspondence Address:
BEYER WEAVER & THOMAS LLP
P.O. BOX 778
BERKELEY, CA 94704-0778 (US)

(73) Assignee: **FLYCODE, Inc.**

(21) Appl. No.: **09/883,064**

(22) Filed: **Jun. 15, 2001**

Related U.S. Application Data

(63) Non-provisional of provisional application No. 60/212,177, filed on Jun. 16, 2000.

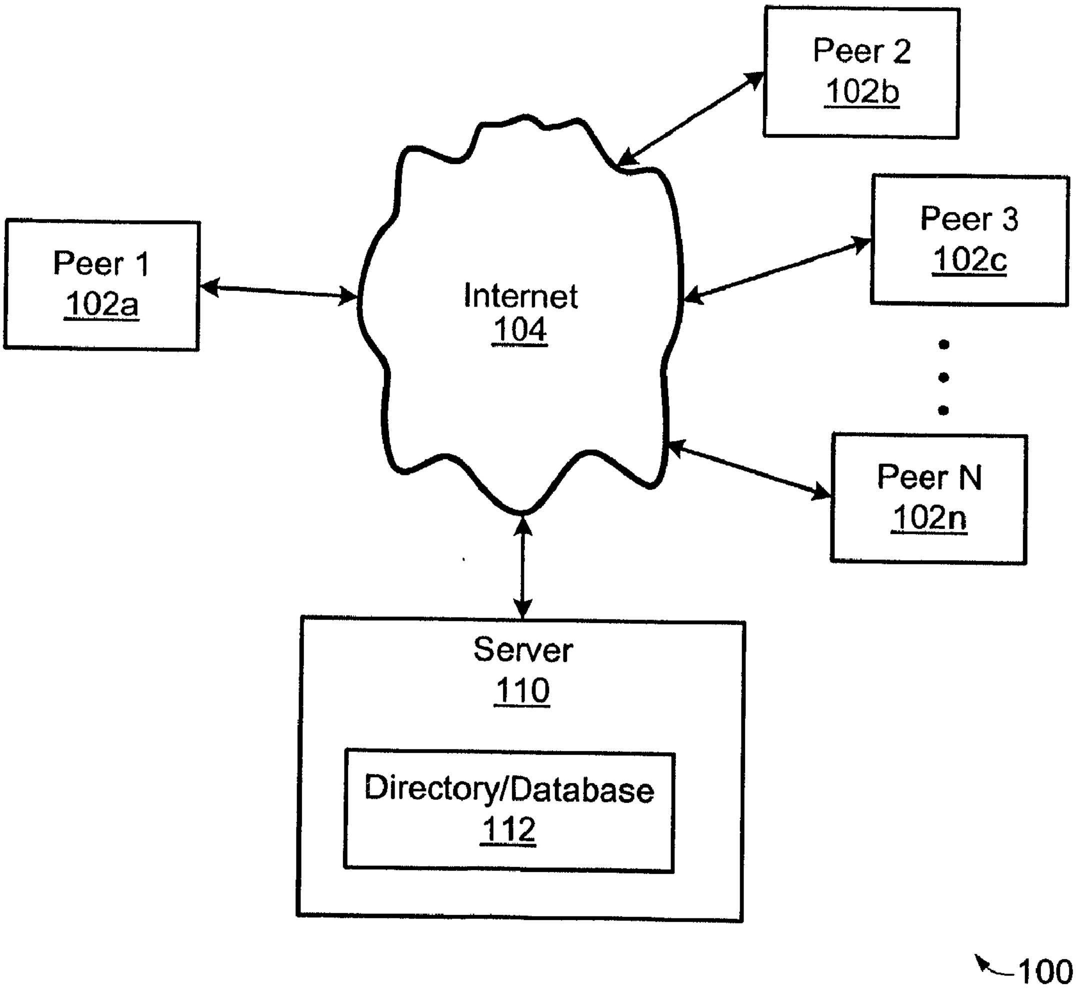
Publication Classification

(51) **Int. Cl.**^{7} **G06F 7/00**

(52) **U.S. Cl.** **707/10**

(57) **ABSTRACT**

The present invention provides an improved technique for accessing information in a peer-to-peer network. According to specific embodiments of the present invention, each file accessible in the peer-to-peer network is assigned a respective hash ID or fingerprint ID which is used to describe the contents of that file. Files in the peer-to-peer network may be identified and/or accessed based upon their associated hash ID values. In this way it is possible to identify identical files stored in the peer-to-peer network which have different file names and/or other metadata descriptors. Since the content of all files having the same hash ID will be identical, an automated process may be used to retrieve the desired content from one or more of the identified files.



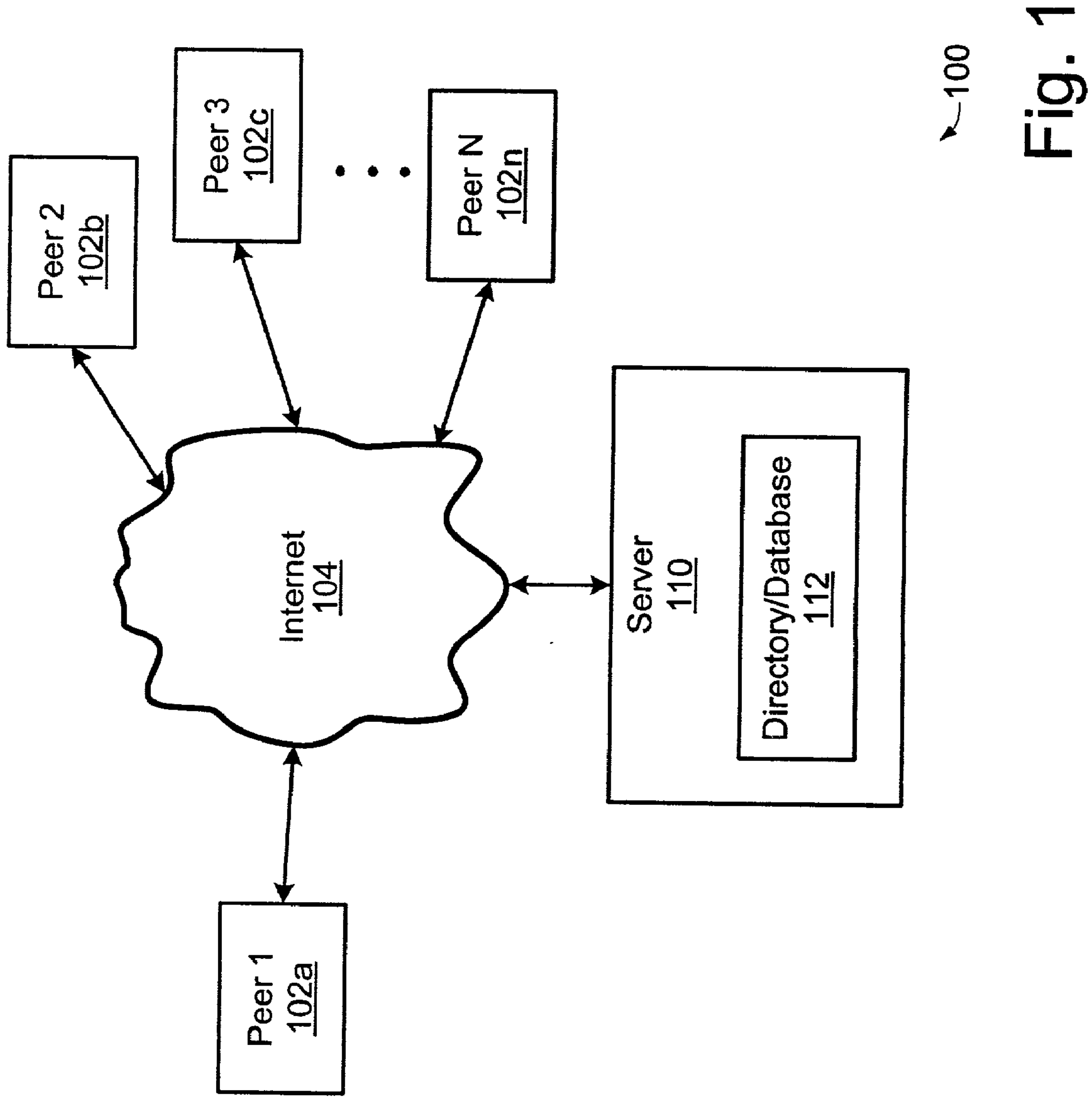

100
Peer 1
102a
Peer 2
102b
Peer 3
102c
Peer N
102n
Internet
104
Server
110
Directory/Database
112

Fig. 1

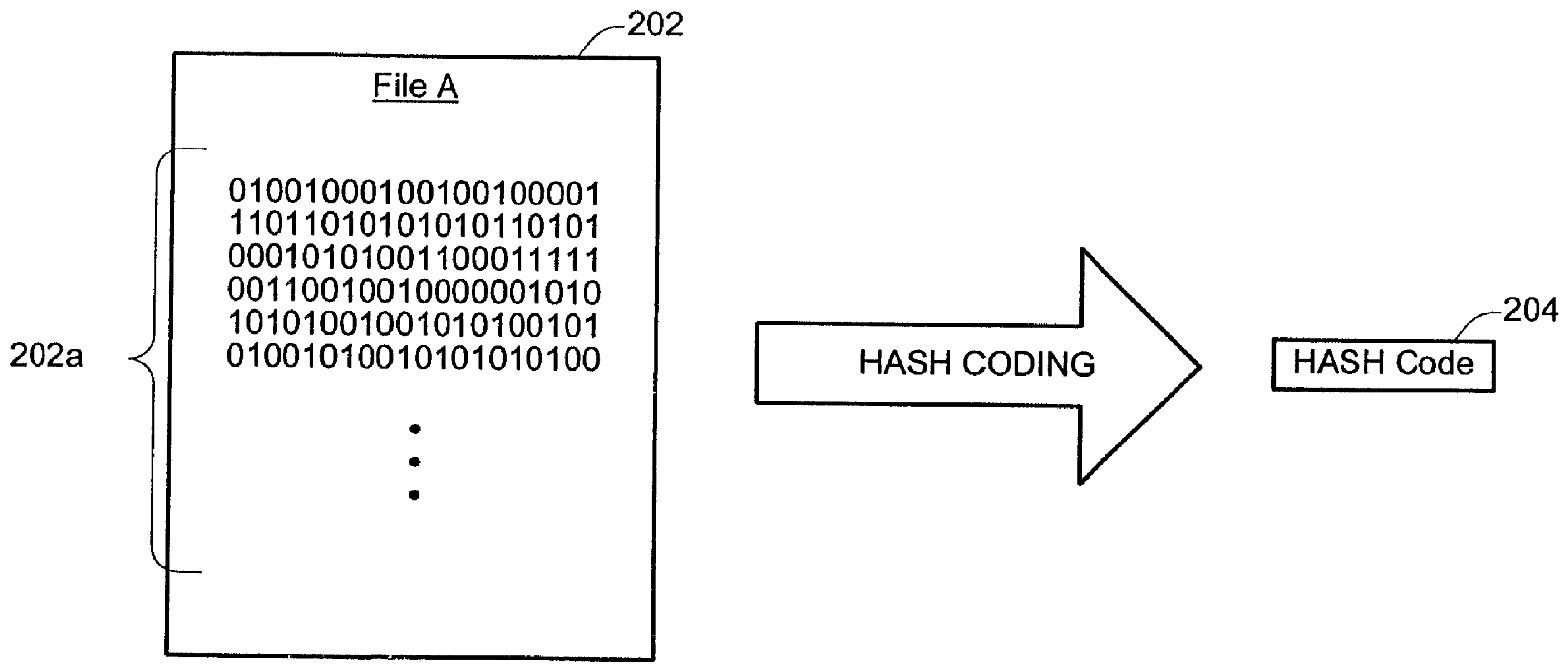

202
File A
01001000100100100001
11011010101010110101
00010101001100011111
00110010010000001010
10101001001010100101
01001010010101010100
• • •
202a
HASH CODING
204
HASH Code

Fig. 2

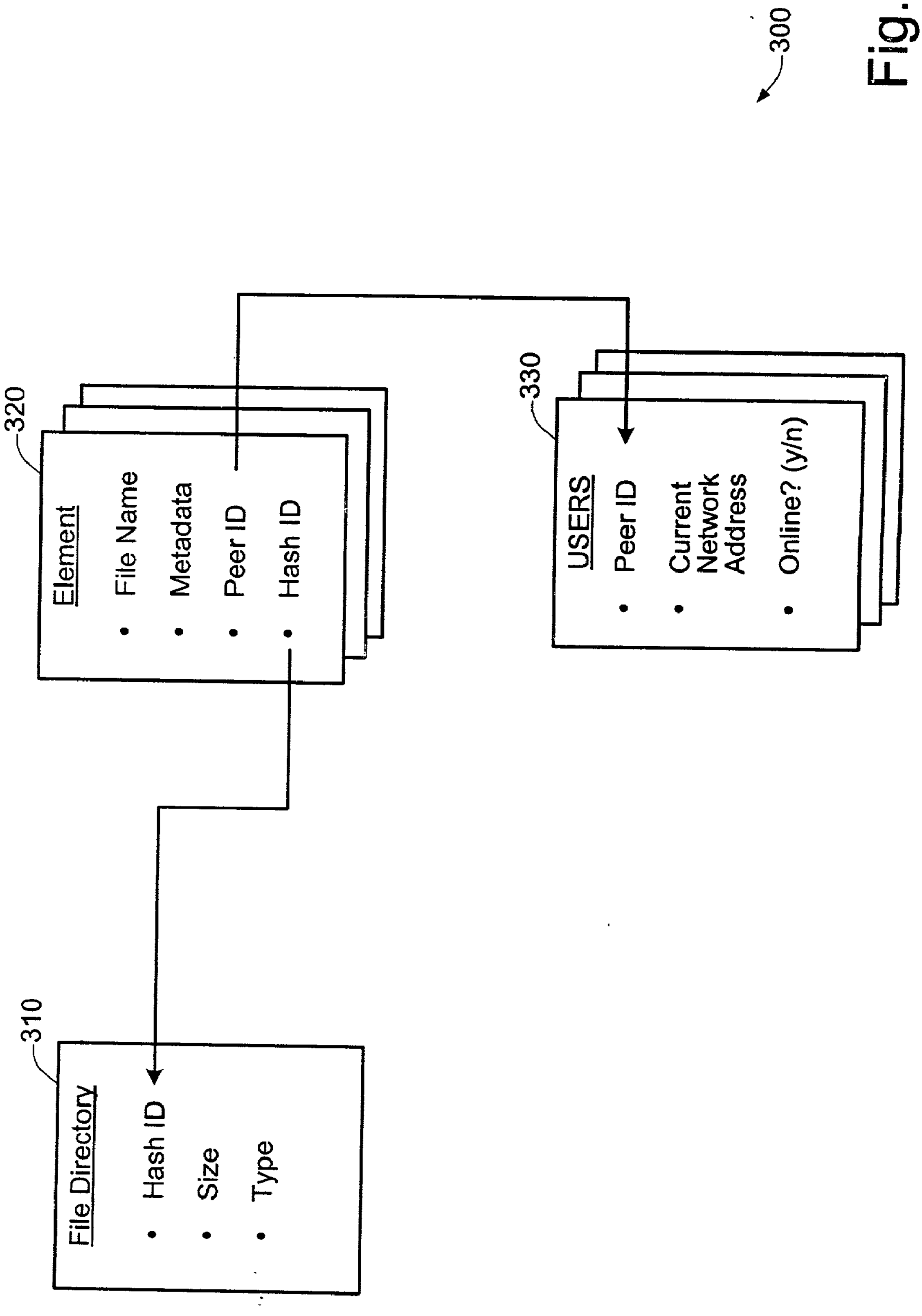
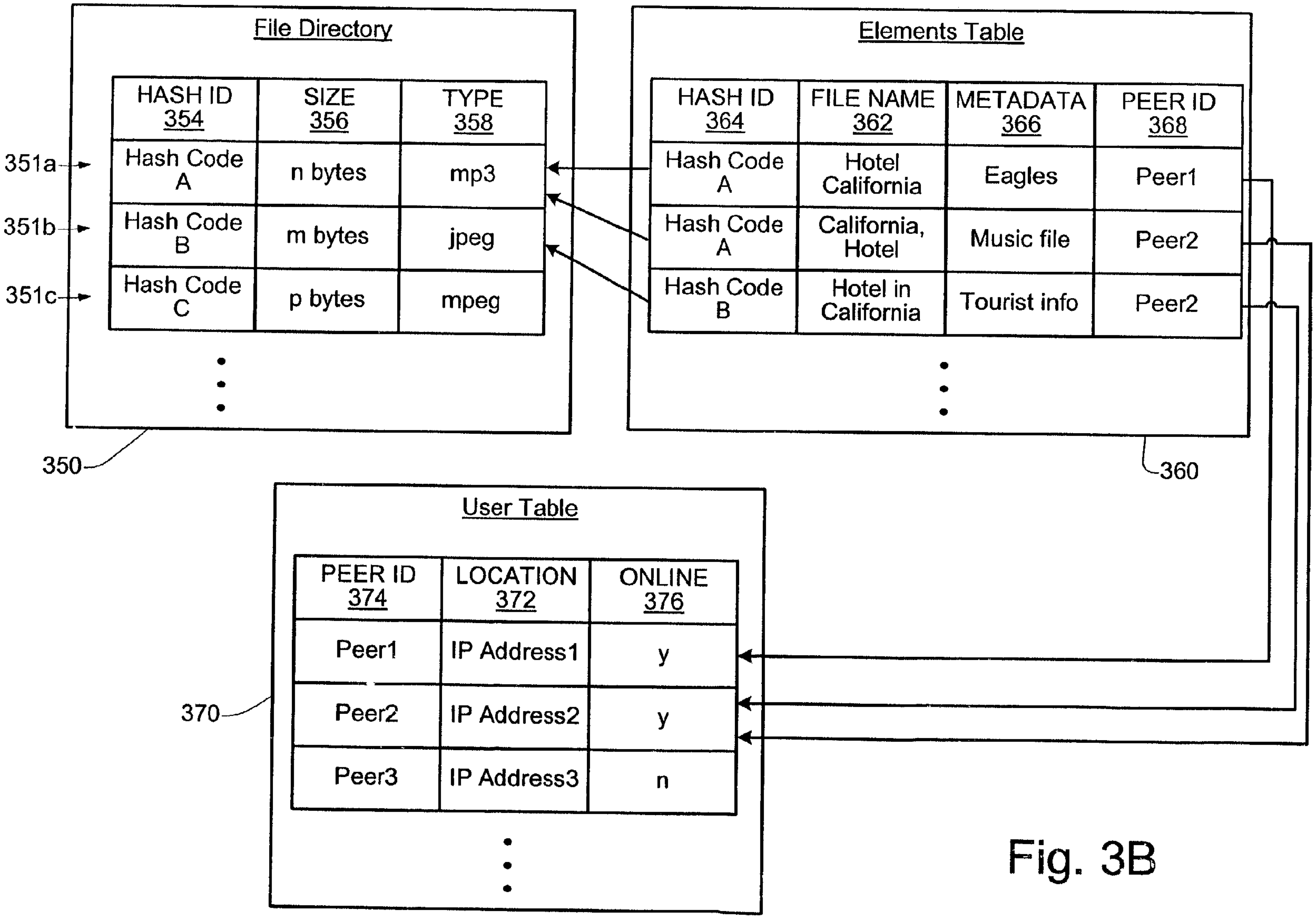

300
320
Element
• File Name
• Metadata
• Peer ID
• Hash ID
330
USERS
• Peer ID
• Current Network Address
• Online? (y/n)
310
File Directory
• Hash ID
• Size
• Type

Fig. 3A



File Directory

HASH ID 354	SIZE 356	TYPE 358
Hash Code A	n bytes	mp3
Hash Code B	m bytes	jpeg
Hash Code C	p bytes	mpeg

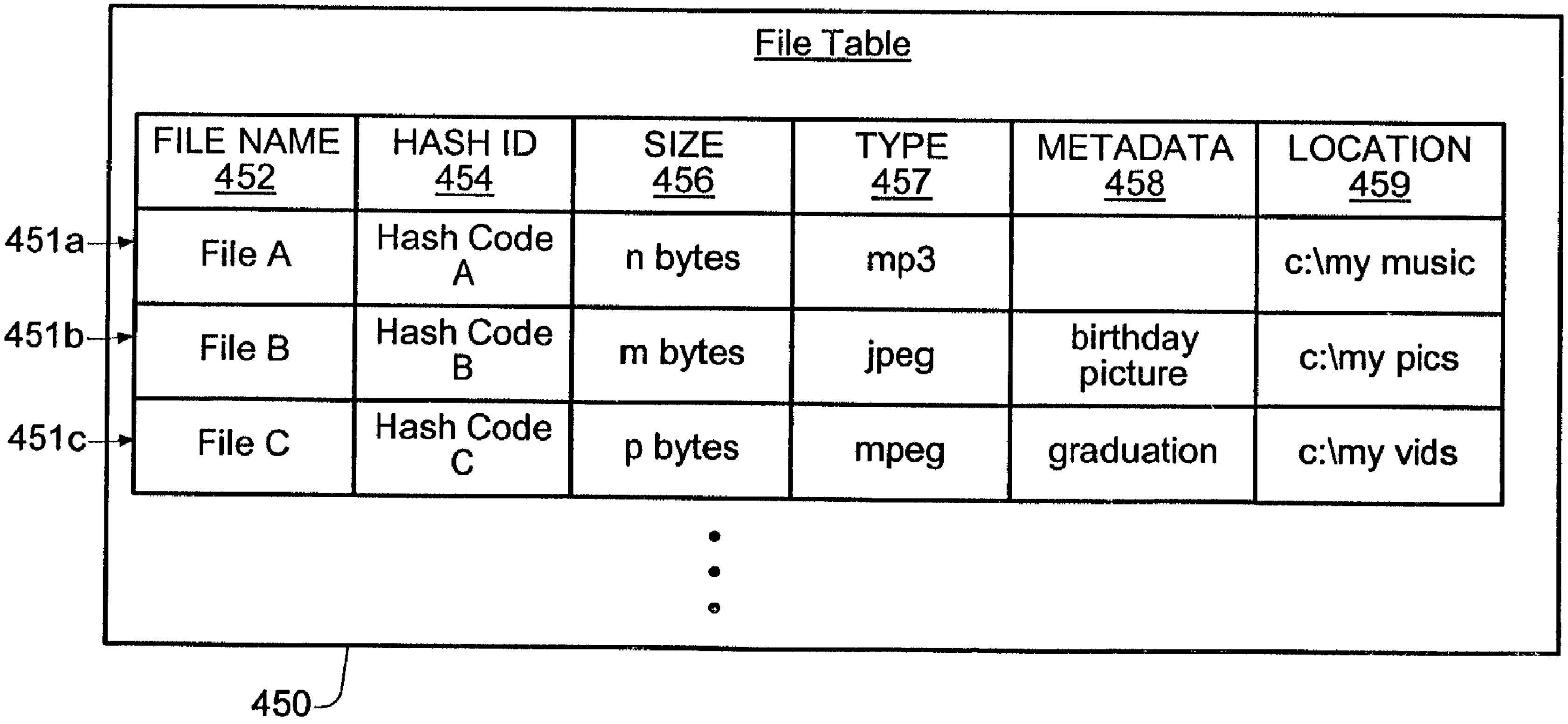
Elements Table

HASH ID 364	FILE NAME 362	METADATA 366	PEER ID 368
Hash Code A	Hotel California	Eagles	Peer1
Hash Code A	California, Hotel	Music file	Peer2
Hash Code B	Hotel in California	Tourist info	Peer2

User Table

PEER ID 374	LOCATION 372	ONLINE 376
Peer1	IP Address1	y
Peer2	IP Address2	y
Peer3	IP Address3	n

Fig. 3B



File Table

FILE NAME 452	HASH ID 454	SIZE 456	TYPE 457	METADATA 458	LOCATION 459
File A	Hash Code A	n bytes	mp3		c:\my music
File B	Hash Code B	m bytes	jpeg	birthday picture	c:\my pics
File C	Hash Code C	p bytes	mpeg	graduation	c:\my vids

451a
451b
451c
450

Fig. 4A

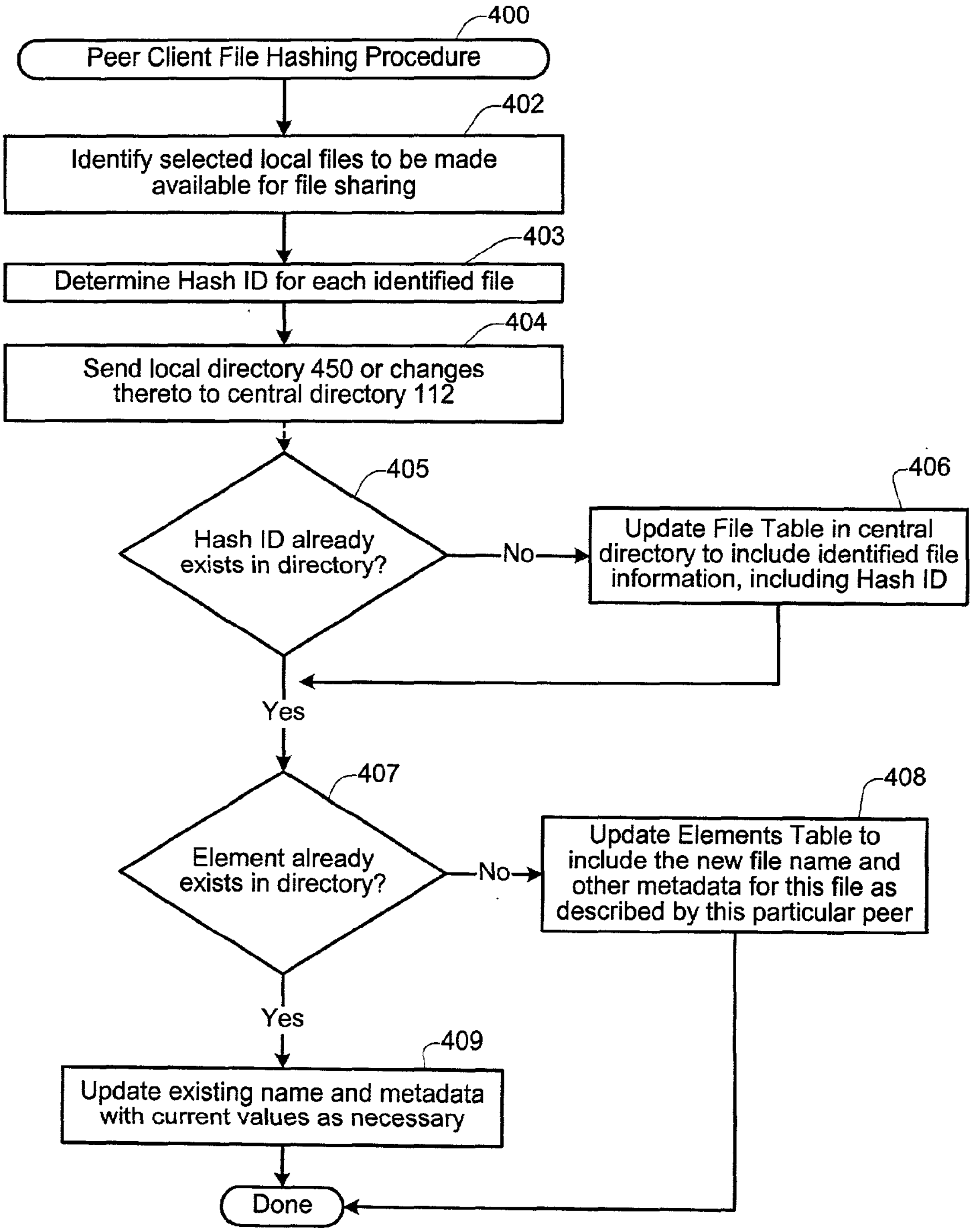

400
Peer Client File Hashing Procedure
402
Identify selected local files to be made available for file sharing
403
Determine Hash ID for each identified file
404
Send local directory 450 or changes thereto to central directory 112
405
Hash ID already exists in directory?
No
406
Update File Table in central directory to include identified file information, including Hash ID
Yes
407
Element already exists in directory?
No
408
Update Elements Table to include the new file name and other metadata for this file as described by this particular peer
Yes
409
Update existing name and metadata with current values as necessary
Done

Fig. 4B

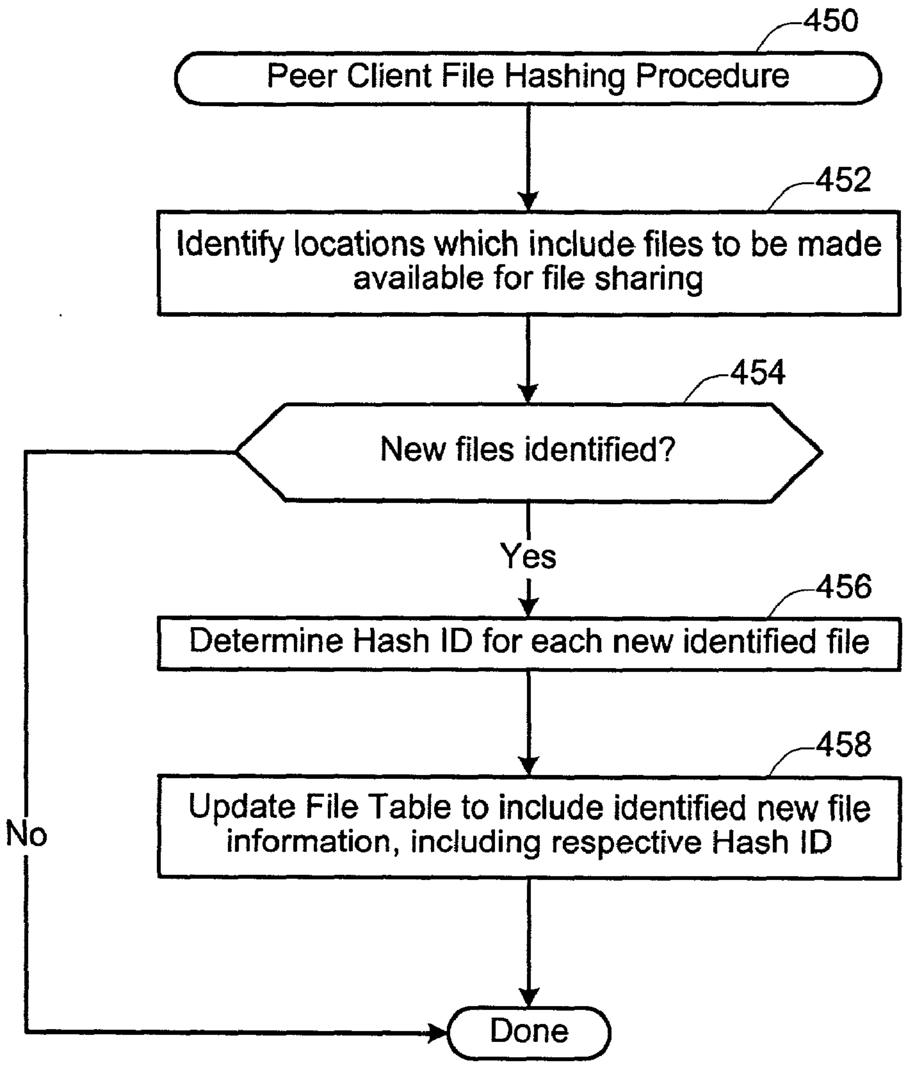

450
Peer Client File Hashing Procedure
452
Identify locations which include files to be made available for file sharing
454
New files identified?
Yes
No
456
Determine Hash ID for each new identified file
458
Update File Table to include identified new file information, including respective Hash ID
Done

Fig. 4C

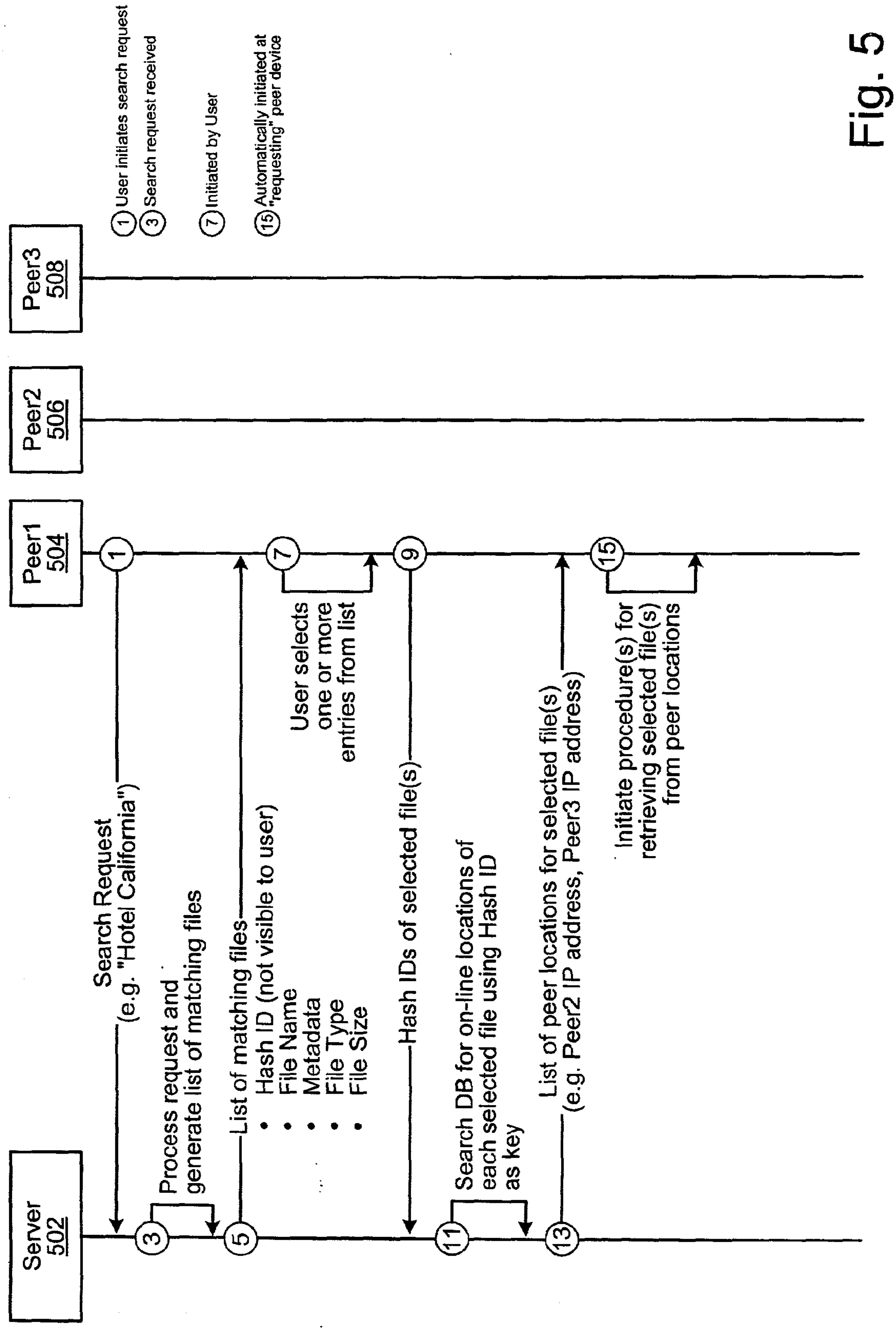
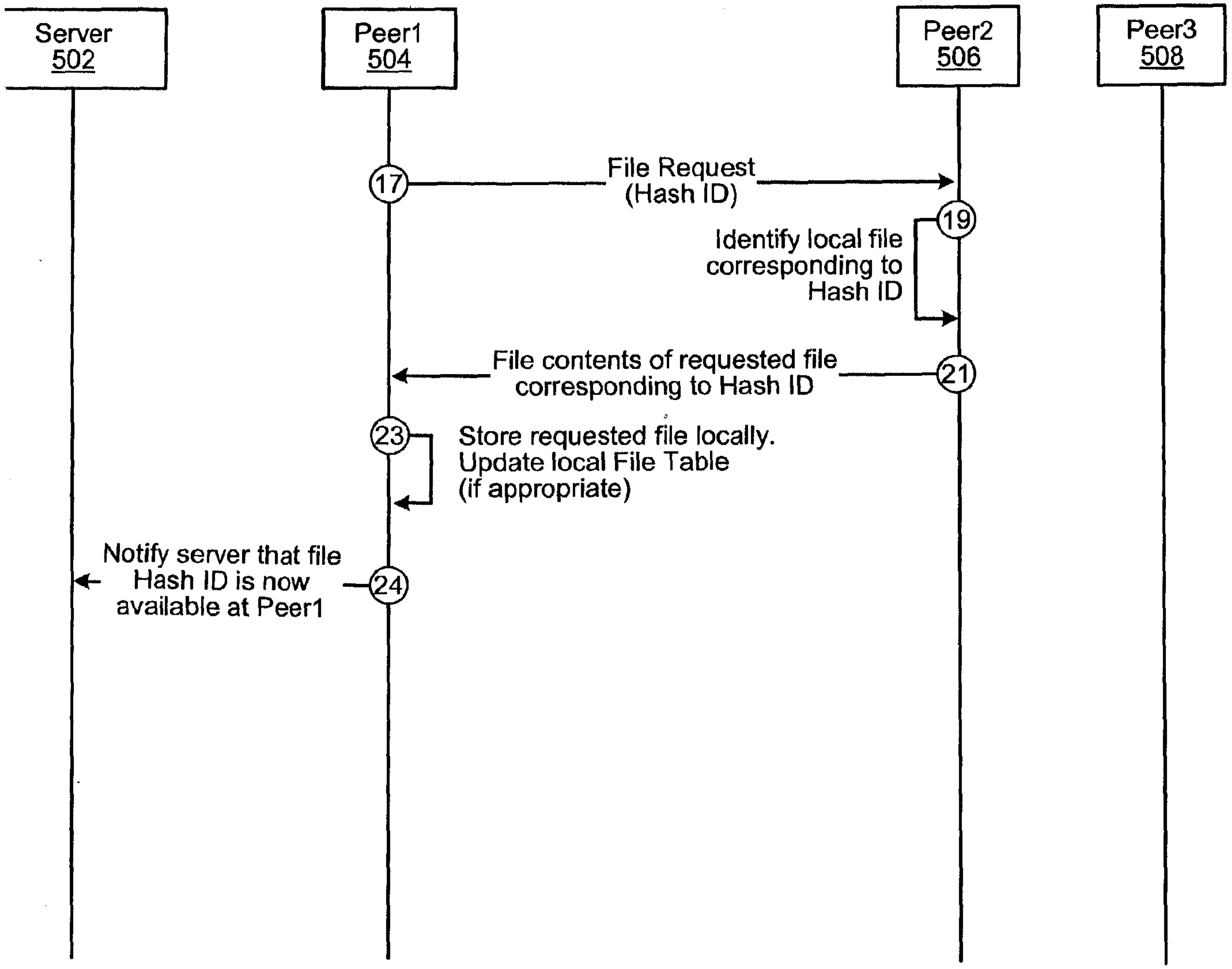

Server 502
Peer1 504
Peer2 506
Peer3 508
1 User initiates search request
3 Search request received
7 Initiated by User
15 Automatically initiated at "requesting" peer device
1
Search Request (e.g. "Hotel California")
3
Process request and generate list of matching files
5
List of matching files
• Hash ID (not visible to user)
• File Name
• Metadata
• File Type
• File Size
7
User selects one or more entries from list
9
Hash IDs of selected file(s)
11
Search DB for on-line locations of each selected file using Hash ID as key
13
List of peer locations for selected file(s) (e.g. Peer2 IP address, Peer3 IP address)
15
Initiate procedure(s) for retrieving selected file(s) from peer locations

Fig. 5



17 Client process at Peer1 automatically intiates file request to selected peer device. Selection of peer device may be based upon:

- Order of file location list
- Random selection from list
- Ping-time
- Proximity factors
- Connection speed
- etc.

Fig. 6A

Round-Robin Embodiment

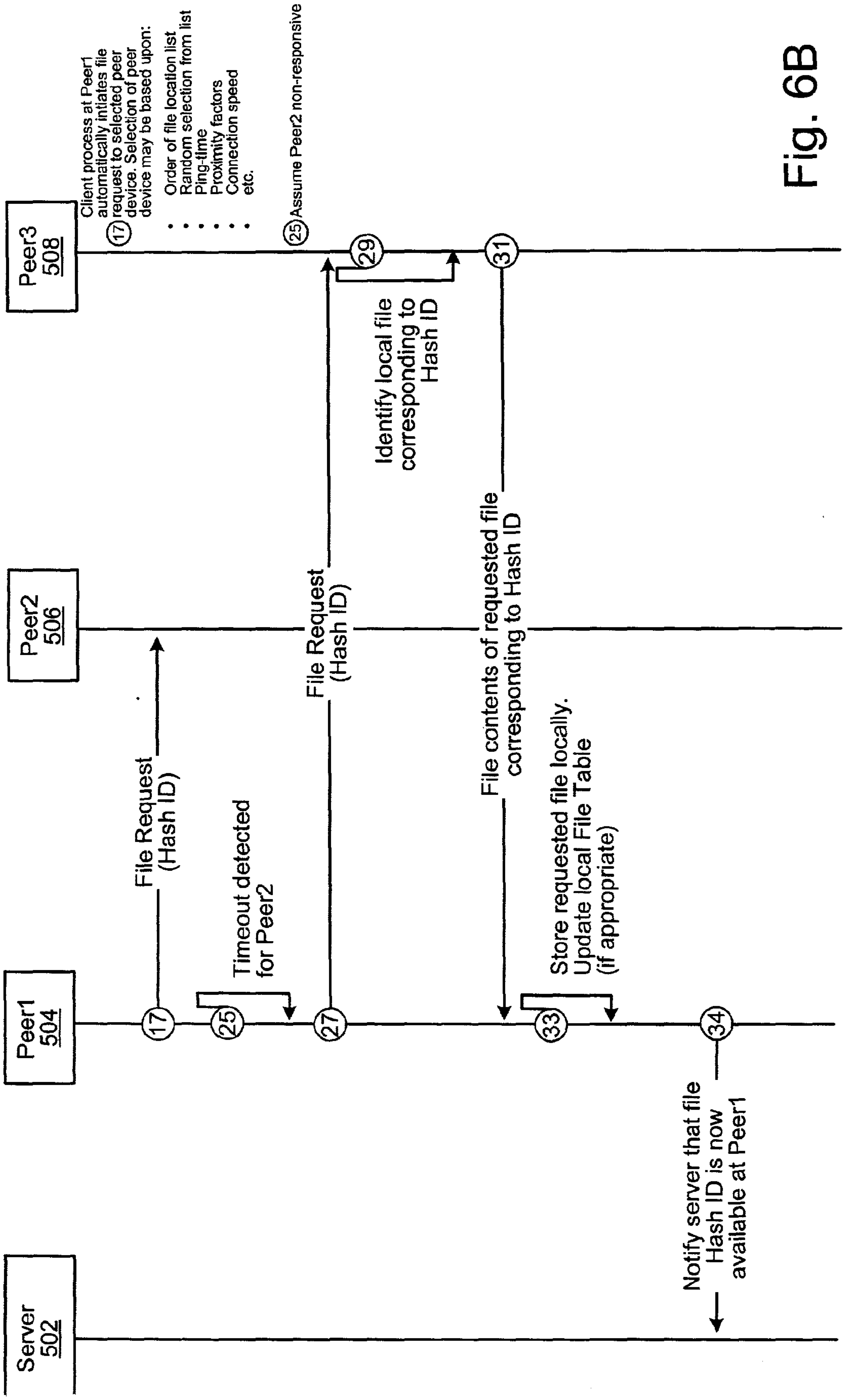

Server
502
Peer1
504
Peer2
506
Peer3
508
Client process at Peer1 automatically intiates file
17 request to selected peer device. Selection of peer device may be based upon:
• Order of file location list
• Random selection from list
• Ping-time
• Proximity factors
• Connection speed
• etc.
25 Assume Peer2 non-responsive
17
File Request (Hash ID)
25
Timeout detected for Peer2
27
File Request (Hash ID)
29
Identify local file corresponding to Hash ID
31
File contents of requested file corresponding to Hash ID
33
Store requested file locally. Update local File Table (if appropriate)
34
Notify server that file Hash ID is now available at Peer1

Fig. 6B

Round-Robin Embodiment

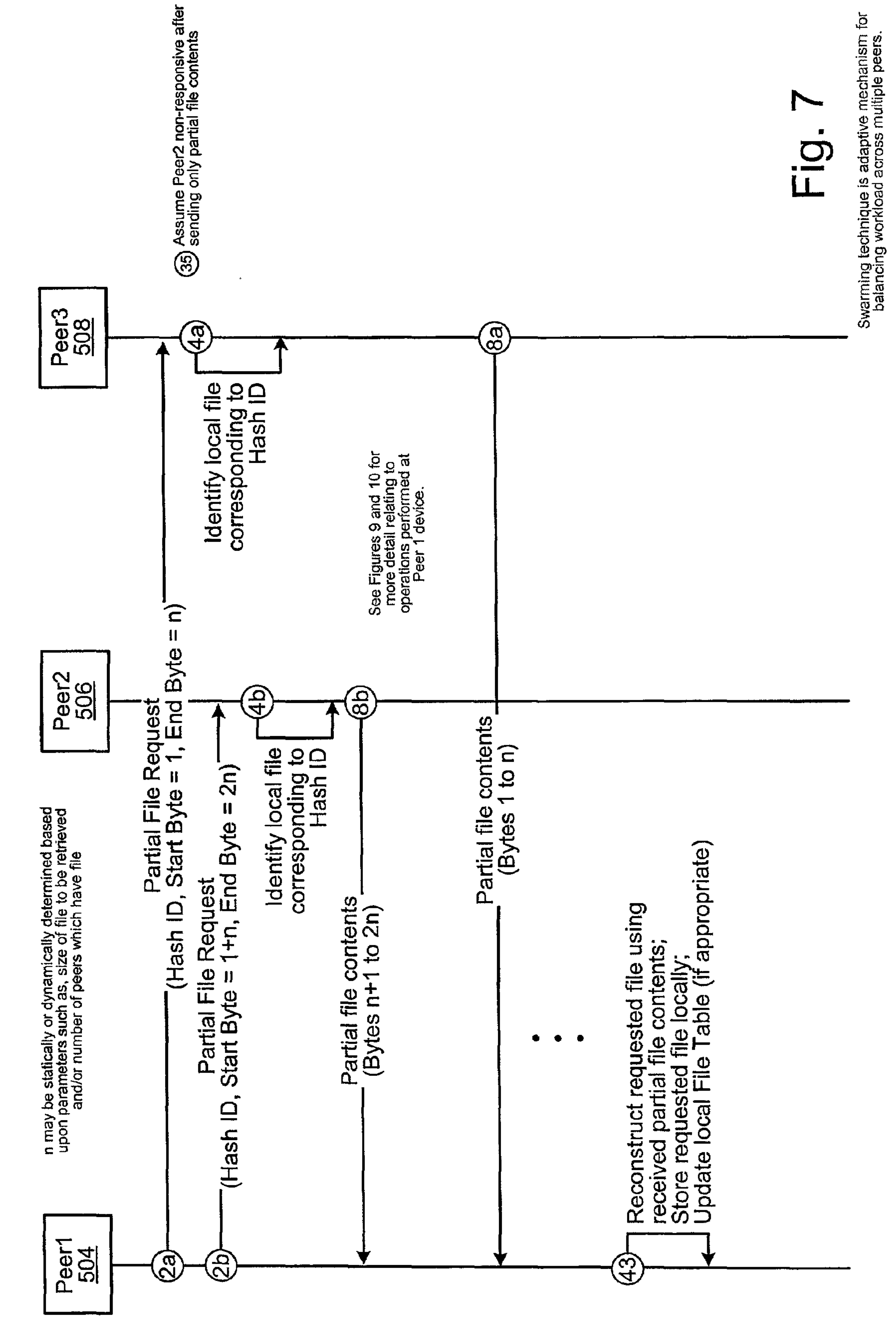
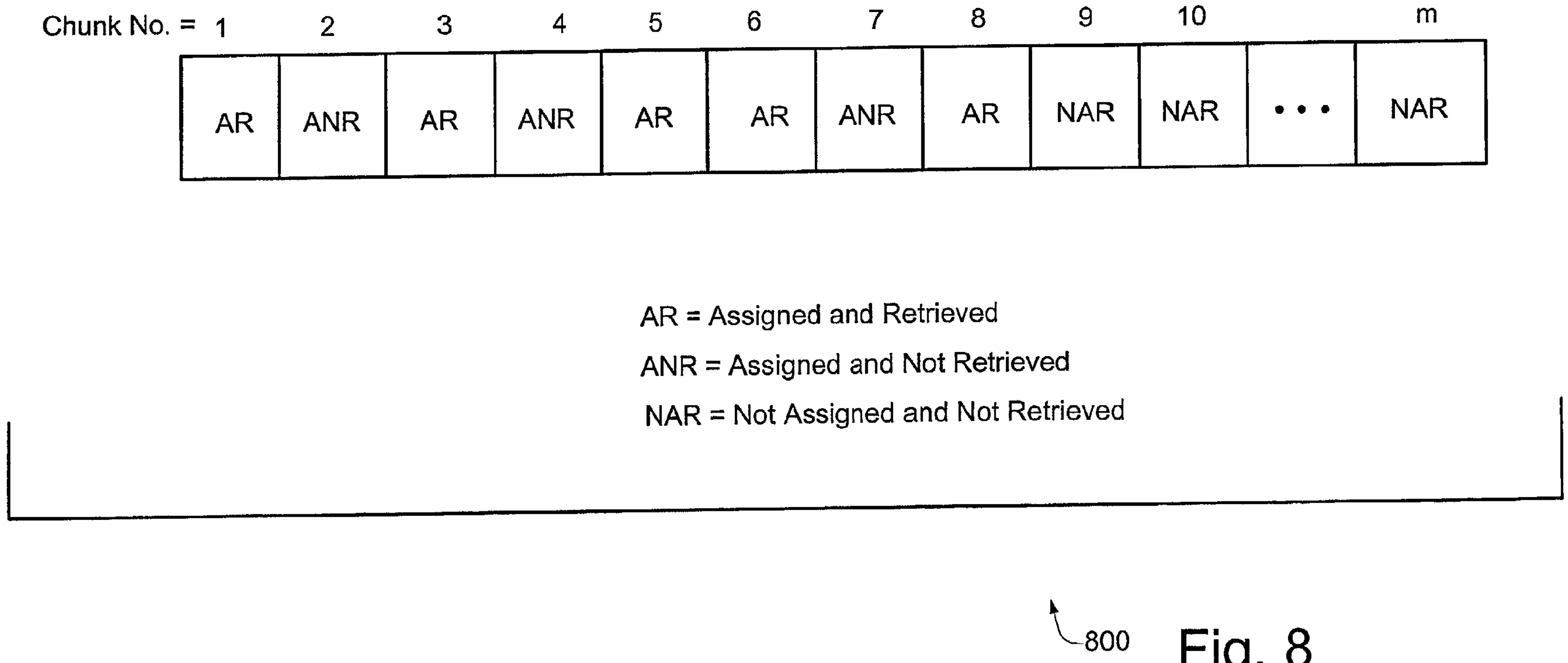

Peer1
504
Peer2
506
Peer3
508
n may be statically or dynamically determined based upon parameters such as, size of file to be retrieved and/or number of peers which have file
2a
Partial File Request
(Hash ID, Start Byte = 1, End Byte = n)
2b
Partial File Request
(Hash ID, Start Byte = 1+n, End Byte = 2n)
35
Assume Peer2 non-responsive after sending only partial file contents
4a
Identify local file corresponding to Hash ID
4b
Identify local file corresponding to Hash ID
See Figures 9 and 10 for more detail relating to operations performed at Peer 1 device.
8b
Partial file contents
(Bytes n+1 to 2n)
8a
Partial file contents
(Bytes 1 to n)
• • •
43
Reconstruct requested file using received partial file contents;
Store requested file locally;
Update local File Table (if appropriate)
Swarming technique is adaptive mechanism for balancing workload across multiple peers.

Fig. 7



Chunk No. =	1	2	3	4	5	6	7	8	9	10		m
	AR	ANR	AR	ANR	AR	AR	ANR	AR	NAR	NAR	• • •	NAR

AR = Assigned and Retrieved

ANR = Assigned and Not Retrieved

NAR = Not Assigned and Not Retrieved

Fig. 8

Chunk Map for desired file.
Maintained by Chunk Manager.

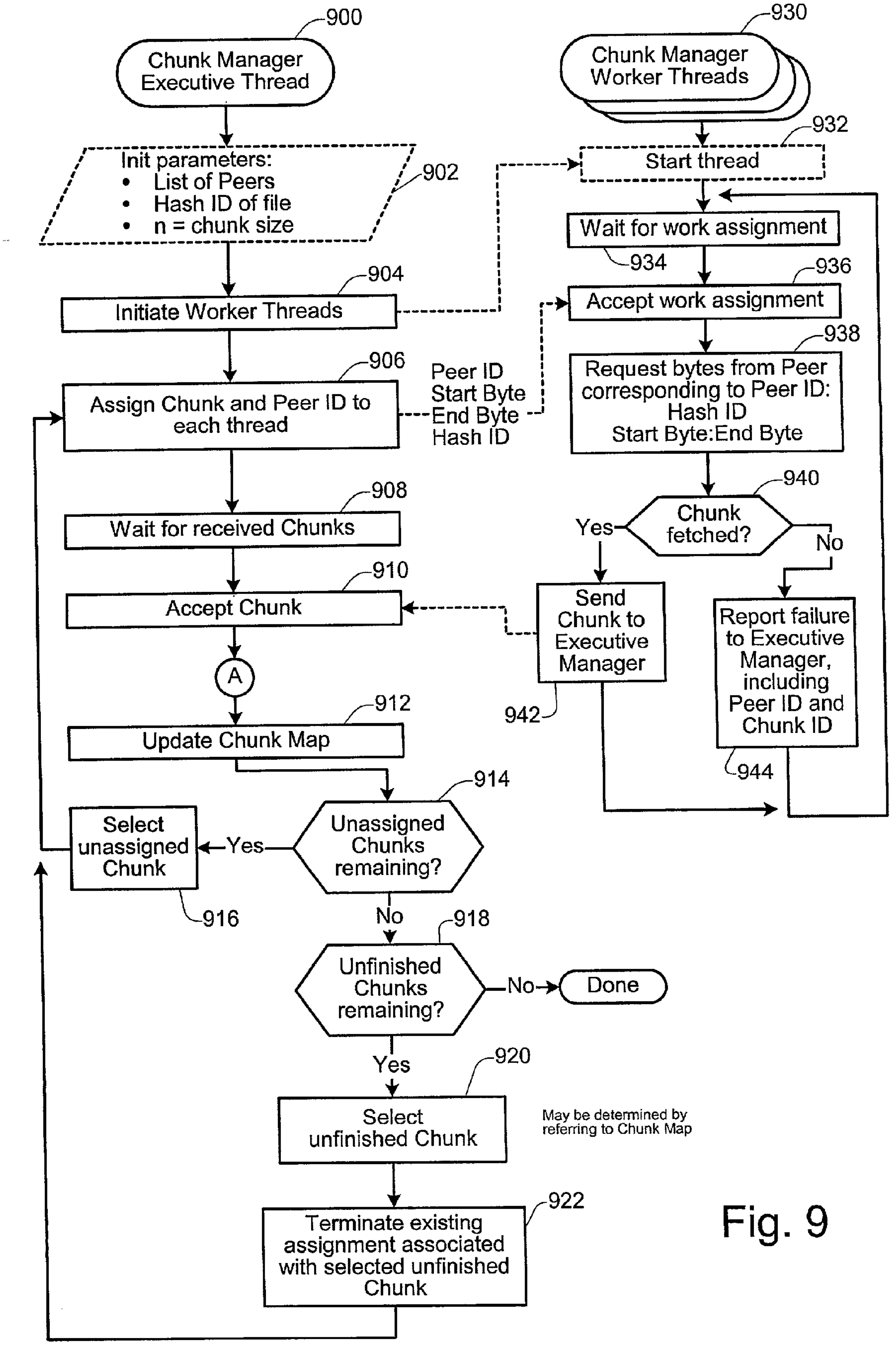

900
Chunk Manager Executive Thread
Init parameters:
• List of Peers
• Hash ID of file
• n = chunk size
902
904
Initiate Worker Threads
906
Assign Chunk and Peer ID to each thread
Peer ID
Start Byte
End Byte
Hash ID
908
Wait for received Chunks
910
Accept Chunk
A
912
Update Chunk Map
914
Unassigned Chunks remaining?
Yes
Select unassigned Chunk
916
No
918
Unfinished Chunks remaining?
No
Done
Yes
920
Select unfinished Chunk
May be determined by referring to Chunk Map
922
Terminate existing assignment associated with selected unfinished Chunk
930
Chunk Manager Worker Threads
932
Start thread
Wait for work assignment
934
936
Accept work assignment
938
Request bytes from Peer corresponding to Peer ID: Hash ID Start Byte:End Byte
940
Chunk fetched?
Yes
No
Send Chunk to Executive Manager
942
Report failure to Executive Manager, including Peer ID and Chunk ID
944

Fig. 9

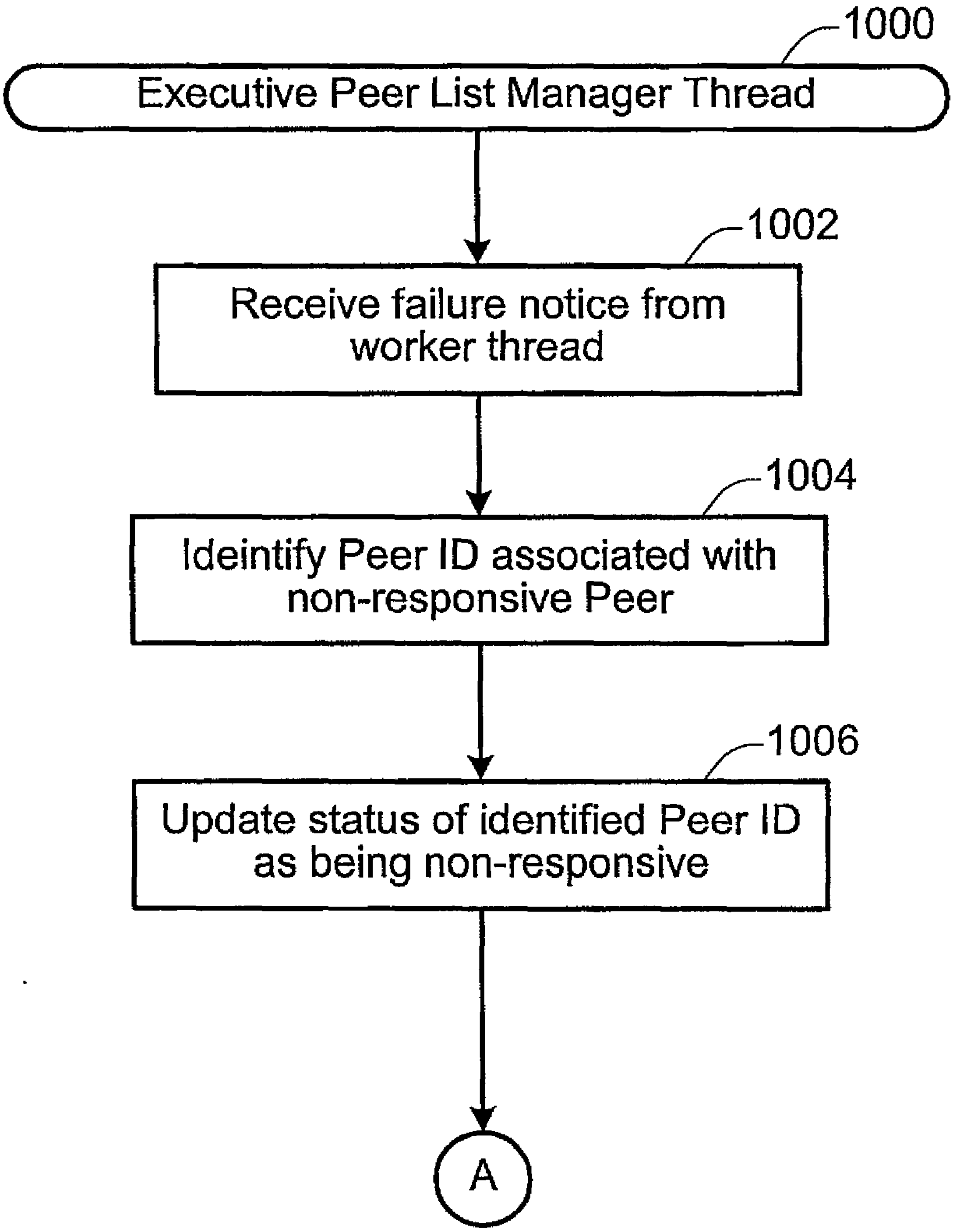

1000
Executive Peer List Manager Thread
1002
Receive failure notice from worker thread
1004
Ideintify Peer ID associated with non-responsive Peer
1006
Update status of identified Peer ID as being non-responsive
A

Fig. 10

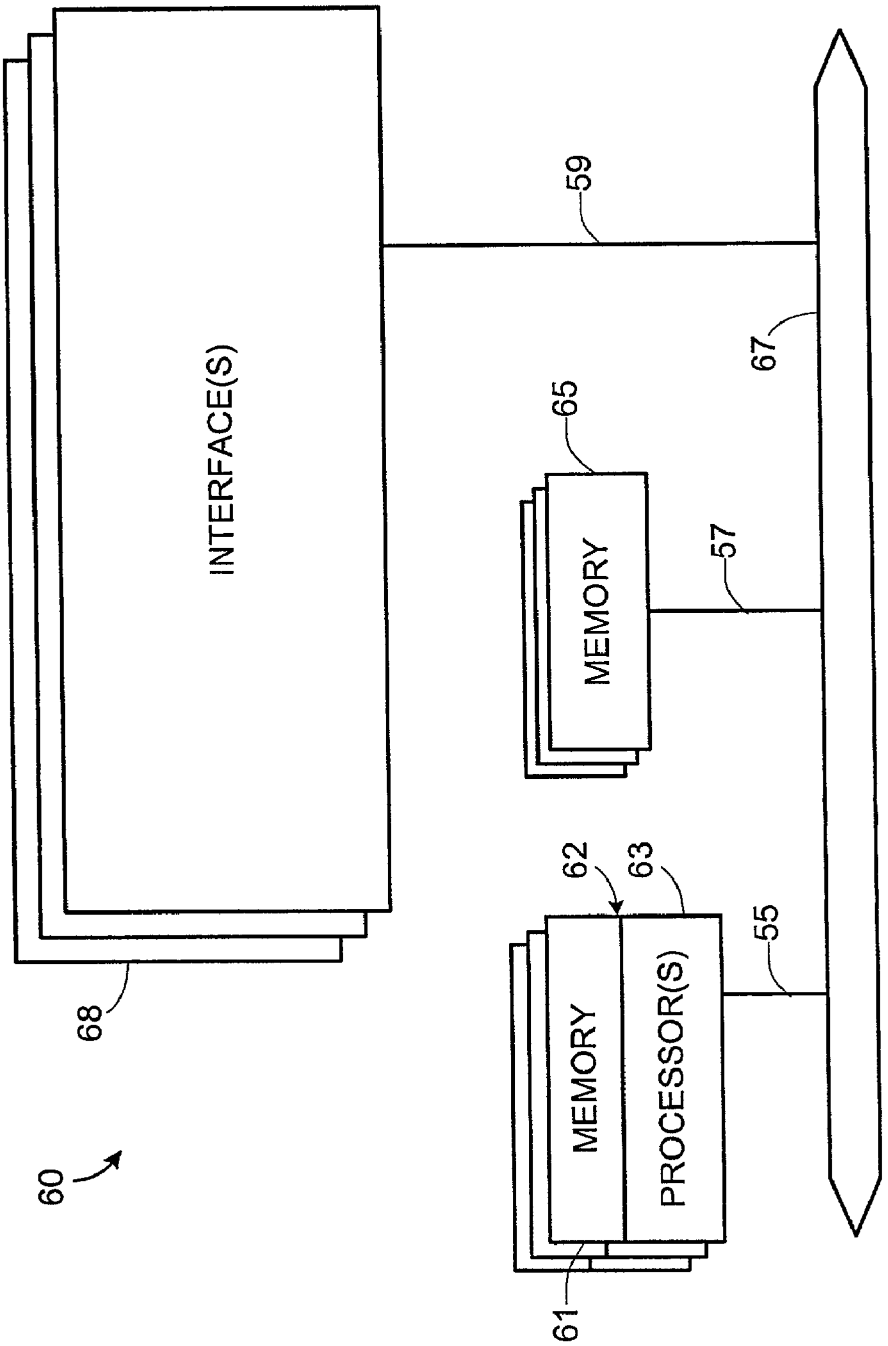
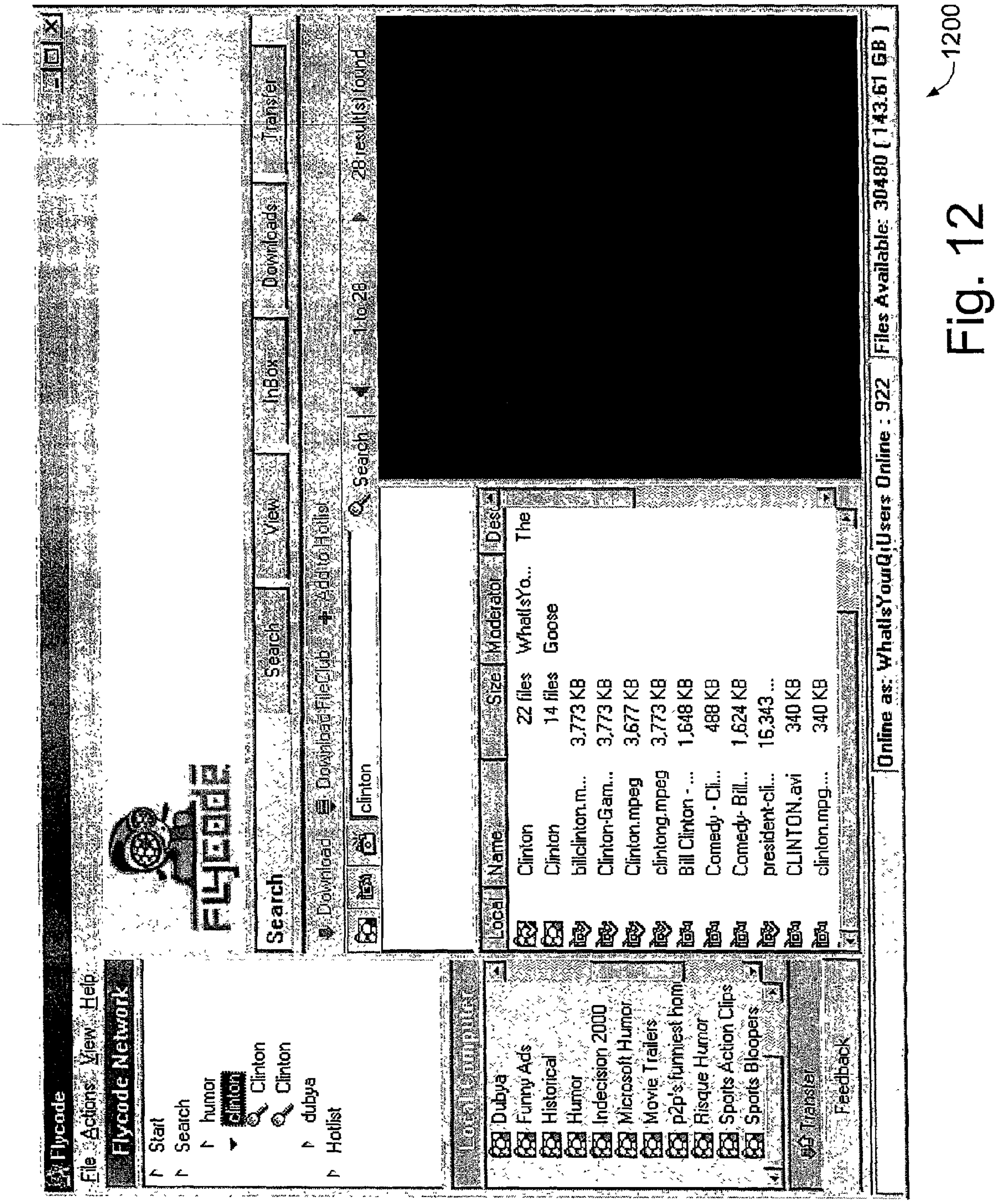

INTERFACE(S)
68
59
67
MEMORY
65
57
MEMORY
PROCESSOR(S)
62
63
55
61
60

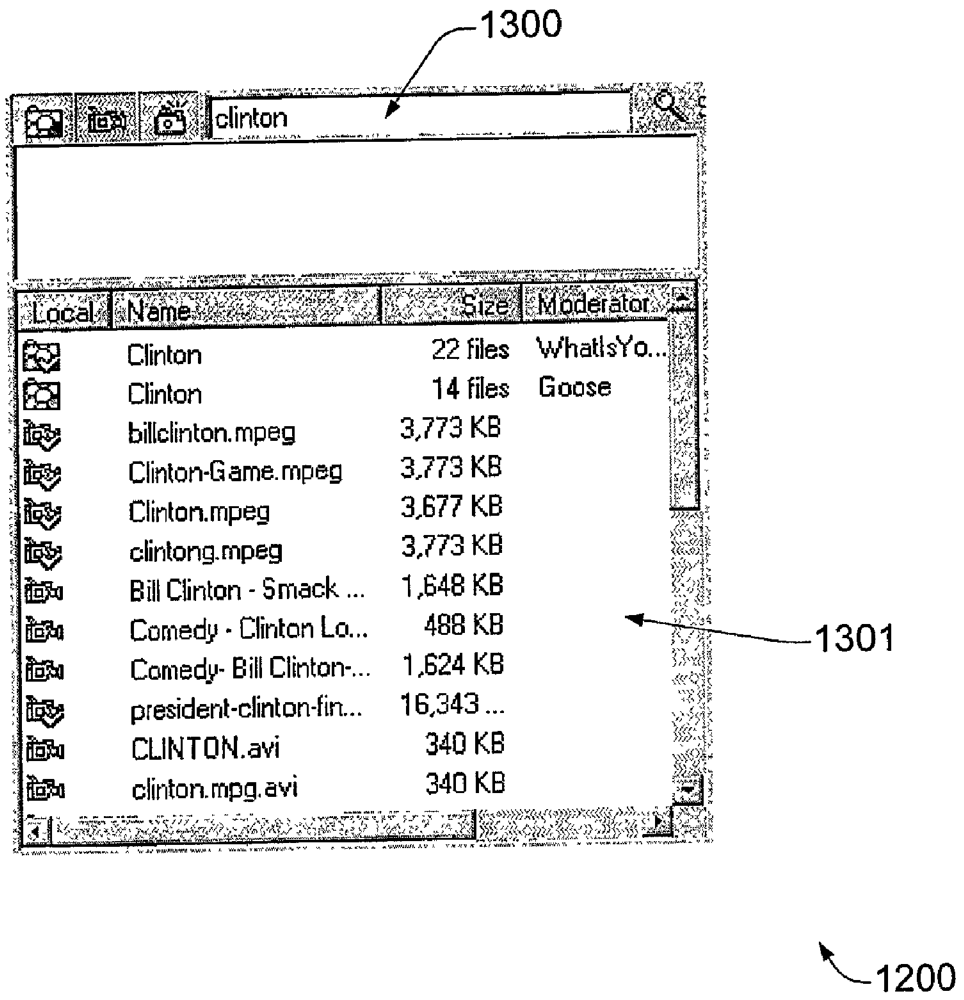
Figure 11



Local	Name	Size	Moderator	Des
	Clinton	22 files	WhatIsYo...	The
	Clinton	14 files	Goose	
	billclinton.m...	3,773 KB		
	Clinton-Gam...	3,773 KB		
	Clinton.mpeg	3,677 KB		
	clintong.mpeg	3,773 KB		
	Bill Clinton - ...	1,648 KB		
	Comedy - Cli...	488 KB		
	Comedy- Bill...	1,624 KB		
	president-cli...	16,343 ...		
	CLINTON.avi	340 KB		
	clinton.mpg....	340 KB		

Online as: WhatIsYourQ | Users Online : 922 | Files Available: 30480 (143.61 GB)

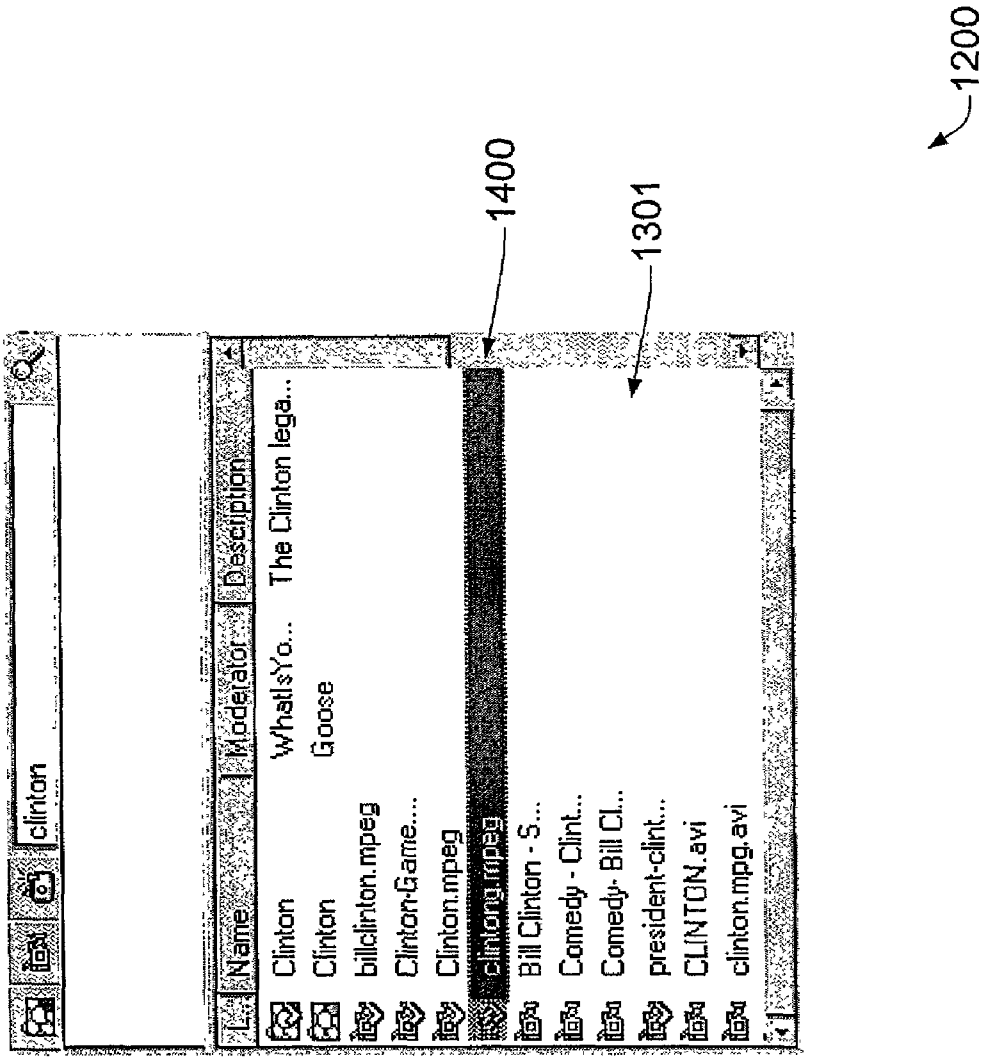
Fig. 12



clinton

Local	Name	Size	Moderator
	Clinton	22 files	WhatIsYo...
	Clinton	14 files	Goose
	billclinton.mpeg	3,773 KB	
	Clinton-Game.mpeg	3,773 KB	
	Clinton.mpeg	3,677 KB	
	clintong.mpeg	3,773 KB	
	Bill Clinton - Smack ...	1,648 KB	
	Comedy - Clinton Lo...	488 KB	
	Comedy- Bill Clinton-...	1,624 KB	
	president-clinton-fin...	16,343 ...	
	CLINTON.avi	340 KB	
	clinton.mpg.avi	340 KB	

Fig. 13



clinton

L...	Name	Moderator	Description
	Clinton	WhatIsYo...	The Clinton lega...
	Clinton	Goose	
	billclinton.mpeg		
	Clinton-Game....		
	Clinton.mpeg		
	clintong.mpeg		
	Bill Clinton - S...		
	Comedy - Clint...		
	Comedy- Bill Cl...		
	president-clint...		
	CLINTON.avi		
	clinton.mpg.avi		

Fig. 14

TECHNIQUE FOR ACCESSING INFORMATION IN A PEER-TO-PEER NETWORK

RELATED APPLICATION DATA

[0001] This application claims priority under 35 U.S.C. Section 119(e) from U.S. Provisional Patent Application No. 60/212,177, filed Jun. 16, 2000, attached hereto as Appendix E, which is incorporated herein by reference in its entirety for all purposes.

BACKGROUND OF THE INVENTION

[0002] Over the past decade, there has been an explosive growth in computer network technology, which has dramatically changed the degree and type of information available to users connected to computer networks, such as, for example, the Internet. As information becomes more accessible over local and wide area networks, new techniques for file storage and distribution are developed. Currently, most existing architectures for file distribution in a network environment utilize centralized file storage and transfer architecture, in which files are stored in central servers and accessed by individual distributed client programs. However, as the files increase in number and size, file storage and distribution from these central servers often becomes problematic.

[0003] One type of file sharing technology which addresses some of the problems posed by centralized file storage systems relates to distributed file storage systems, such as those implemented in peer-to-peer networks. As commonly known to one having ordinary skill in the art peer-to-peer networks may be used for implementing distributed file sharing systems wherein selected files stored on each peer network device may be made accessible to other peer network devices in the peer-to-peer network. Accordingly, peer-to-peer network architectures are highly scalable, since files may be retrieved from many locations rather than just one central location (e.g., a central server).

[0004] In recent years, there have been significant advances in peer-to-peer network technology, particularly with regard to the Internet. For example, peer-to-peer file sharing applications such as NAPSTER™ and GNUTELLA™ now provide the ability for Internet users to configure their computer systems to function as peer network devices in a peer-to-peer network implemented across the Internet. In this way, an Internet user is able to access desired files which are stored at the computer systems of other Internet users.

[0005] While this first generation peer-to-peer architecture solved some of the problems associated with centralized file storage, it also introduced new problems such as, for example, file access, reliability, speed, security, etc. For example, using peer-to-peer file sharing applications such as NAPSTER™, shared files in the peer-to-peer network were identified and retrieved based upon their file names. Thus, for example, if a name of a file were misspelled, there was no other way of identifying the file during a search. Additionally, if a peer which was currently involved in one or more file retrieval operations went offline, the file retrieval operations would fail. The requesting user then had to discard the partial file contents and pick a new peer to download from. Consequently, very large files were virtually impossible to retrieve since few peers remained online long enough to complete such a large transfer.

[0006] It will be appreciated that there are numerous issues relating to peer-to-peer network technology which remain to be resolved. Accordingly, continuous efforts are being undertaken to improve peer-to-peer networking technology in order to provide improved file storage, access, and distribution techniques implemented over a data network.

SUMMARY OF THE INVENTION

[0007] According to different embodiments of the present invention, methods, systems, and computer program products are disclosed for accessing information in a peer-to-peer network. The peer-to-peer network includes a plurality of peer devices and a database accessible by at least a portion of the peer devices. Each of the peer devices is configured to store information files, and is further configured to share content from selected information files with at least a portion of the other peer devices in the network. Each shared file in the network has a respective fingerprint IID associated therewith relating to its file content.

[0008] According to specific embodiments, files in the peer-to-peer network may be identified and/or accessed based upon their associated hash ID values. In this way it is possible to identify identical files stored in the peer-to-peer network which have different file names and/or other metadata descriptors. Additionally, since the content of all files having the same hash ID will be identical, an automated process may be used to retrieve the desired content from one or more of the identified files. For example, a user may elect to retrieve a desired file (having an associated hash ID) which may be stored at one or more remote locations in the peer-to-peer network. Rather than the user having to select a specific location for accessing and retrieving the desired file, an automated process may use the hash ID (associated with the desired file) to automatically select one or more remote locations for retrieving the desired file. According to different embodiments, the automated process may choose to retrieve the entire file contents of the desired file from a specific remote location, or may choose to receive selected portions of the file contents of the desired file from different remote locations in the peer-to-peer network. Further, if an error occurs during the file transfer process, resulting in a partial file transfer, the automated process may be configured to identify the portion(s) of the desired file which were not retrieve, and automatically select at least one different remote location for retrieving the remaining contents of the desired file.

[0009] Additional objects, features and advantages of the various aspects of the present invention will become apparent from the following description of its preferred embodiments, which description should be taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] **FIG. 1** illustrates a block diagram of peer-to-peer network which may be used for implementing the technique of the present invention in accordance with a specific embodiment of the present invention.

[0011] **FIG. 2** shows a block diagram of HASH coding of a file in accordance with a specific embodiment of the present invention.

[**0012**] **FIG. 3A** shows a block diagram of directory data structures in accordance with a specific embodiment of the present invention.

[**0013**] **FIG. 3B** shows a block diagram of an example of specific data stored in the directory data structures in accordance with a specific embodiment of the present invention.

[**0014**] **FIG. 4A** shows a block diagram of an example of specific data stored in the peer directory structures in accordance with a specific embodiment of the present invention.

[**0015**] **FIGS. 4B and 4C** illustrates flow diagrams of the directory synchronization process between a local peer directory and a central directory in accordance with a specific embodiment of the present invention.

[**0016**] **FIG. 5** illustrates a trace diagram of the technique for searching files in accordance with a specific embodiment of the present invention.

[**0017**] FIGS. 6A-6C illustrates a trace diagram of a file retrieving technique in accordance with a specific embodiment of the present invention.

[**0018**] **FIG. 7** shows a trace diagram of another file retrieving technique from multiple peers in accordance with an alternative embodiment of the present invention.

[**0019**] **FIG. 8** shows a block diagram of a chunk map for the management of the retrieval of "chunks" of a file for the file retrieving technique in accordance with the alternative embodiment of the present invention.

[**0020**] **FIG. 9** illustrates a flow diagram of the chunk management technique across multiple worker threads for the file retrieving technique in accordance with the alternative embodiment of the present invention.

[**0021**] **FIG. 10** illustrates a flow diagram of the chunk management technique when an unresponsive peer for the file retrieving technique in accordance with the alternative embodiment of the present invention.

[**0022**] **FIG. 11** shows a specific embodiment of a peer network device **60** which may be used for implementing the technique of the present invention.

[**0023**] **FIG. 12** is a diagram of an example of a screen shot illustrating a user interface on a peer device in accordance with a specific embodiment of the present invention.

[**0024**] **FIG. 13** is a diagram of another example of a screen shot illustrating a user interface showing a search input field in accordance with a specific embodiment of the present invention.

[**0025**] **FIG. 14** is a diagram of the example of a screen shot of **FIG. 13** illustrating a selection of the search input field in accordance with a specific embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[**0026**] The present invention provides an improved technique for accessing information in a peer-to-peer network. According to specific embodiments of the present invention, each file accessible in the peer-to-peer network is assigned a respective hash ID or fingerprint ID which is used to describe the contents of that file. According to one embodiment, a conventional hash or fingerprinting algorithm may be used to analyze the contents of a selected file, and generate a unique hash ID or fingerprint ID which may be used for identifying the specific contents of that file. The hashing algorithm is designed such that no two files having different file content will have the same hash ID. However, files having identical file content will have the same hash ID. In one implementation, the file name and metadata associated with a file are not included in the computation of the hash ID for that file.

[**0027**] According to specific embodiments, files in the peer-to-peer network may be identified and/or accessed based upon their associated hash ID values. In this way it is possible to identify identical files stored in the peer-to-peer network which have different file names and/or other metadata descriptors. Additionally, since the content of all files having the same hash ID will be identical, an automated process may be used to retrieve the desired content from one or more of the identified files. For example, a user may elect to retrieve a desired file (having an associated hash ID) which may be stored at one or more remote locations in the peer-to-peer network. Rather than the user having to select a specific location for accessing and retrieving the desired file, an automated process may use the hash ID (associated with the desired file) to automatically select one or more remote locations for retrieving the desired file. According to different embodiments, the automated process may choose to retrieve the entire file contents of the desired file from a specific remote location, or may choose to receive selected portions of the file contents of the desired file from different remote locations in the peer-to-peer network. Further, if an error occurs during the file transfer process, resulting in a partial file transfer, the automated process may be configured to identify the portion(s) of the desired file which were not retrieve, and automatically select at least one different remote location for retrieving the remaining contents of the desired file.

[**0028**] Referring now to **FIG. 1, a** high level view of a peer-to-peer network **100** is illustrated in accordance with a specific embodiment of the present invention. The network **100** includes a plurality of peer network devices **102***a*-**102***n*, and at least one central server system **110**. According to one specific implementation, the peer devices **102***a*-**102***n* are communicably connected to each other and to the central server **110** via the Internet **104**. The peers may communicate with each other and the server via the http protocol or via a private protocol, the former being preferable to minimize the effects of various firewalls that may exist between any given peer device and the Internet **104**.

[**0029**] The server **110** preferably includes software or firmware that handles communication between the peer devices **102***a*-**102***n* and the central server **110**, and that performs specific logical operations on the directory **112** on the server. According to one specific implementation, the directory **112** may be stored in a single relational database, but permanent storage may also be accomplished by an object database, a directory server, or multiple databases.

[**0030**] Files **202** that are to be shared are only stored on the peer devices **102***a*-**102***n*. The directory **112** stores information about the files (e.g., HASH ID, filename, metadata, size, type, etc.) but not their contents. In addition, the directory **112** stores information about the peer devices themselves (e.g., Peer ID) and most importantly, which peer devices the files **202** exist on at any given time.

[**0031**] Briefly, it will be appreciated that file **202** may contain any type of stored information. These include, for example, jpeg, mpeg and mp3.

[**0032**] Turning now to **FIG. 2**, the unique footprinting of a file **202** is illustrated in accordance with one specific embodiment of the present invention. In particular, the process by which a file **202** may be uniquely identified is through the application of HASH code **204**. Each peer device **102***a*-**102***n* that wishes to publish a file on the peer-to-peer network **100** first computes the HASH code **204** of that file. The HASH code **204** may be computed via conventional algorithms which generate a unique identifier based upon the content of a particular file. Examples of fingerprinting or hash code algorithm which may be used in conjunction with the technique of the present invention include, MD5 (described in RFC 1321, and attached hereto has Appendix A), and Keyed SHAL (described in RFC 2841, and attached hereto has Appendix B). Each of these references is incorporated herein by reference in its entirety for all purposes.

[**0033**] According to one specific implementation, MD5 (See Appendix A) may be used to guarantee that the HASH code is unique for each different file, even when two files differ by as little as one bit. Other algorithms may also be used such as, for example, Keyed SHAI (See Appendix B) which mathematically characterizes the audio or visual waveforms of the content of the file and creates a unique code representation of that file. According to a specific embodiment, the filename, time/date stamps, and any other meta-data about the file are not included in the computation of the HASH code. Different peers, or even the same peer, can refer to or describe what is in fact the same file content by different names. The desired files are more likely to be found, as different users may search for files using different names, descriptions, or other characteristics, although the file content is identical. Such unique ID association allows each user to name and describe a given file in the manner most relevant to them.

[**0034**] According to one specific implementation, the central directory data **300** may be stored in a relational database. **FIG. 3A** illustrates three of the central tables in the directory data structures **300**: the file directory table **310**; the elements table **320**; and the users table **330**. The file directory table **310**, for example, contains information about the file contents only, including the HASH code or ID, the size, and the type (video, music, image, etc . . .). Other information specific to the contents of the file may also be stored in this table. The elements table **320**, on the other hand, contains descriptive information or meta data about each file as entered and maintained by each individual user. This elements table **320** also includes foreign keys Hash code ID and Peer device ID that points to the files **310** and users tables **330**, respectively. The users table **330** contains information about each user and his/her peer device **102***a*-**102***n*. The users table also maintains the systems current knowledge as to which user is currently connected to the peer-to-peer network **100**.

[**0035**] According to one specific embodiment, the relationship between the three tables of **FIG. 3A** can be defined such that each user can have many files and each file can be owned by many users. In addition, each user can have their own individual description of each file and may possibly describe the same file in more than one way, although the file content, and thus, the HASH coding are identical.

[**0036**] In one embodiment, should a file be deleted by every user that ever had them, there will be no record in the elements table **320** of that file but a record of that file's existence will remain in the file directory table **310**.

[**0037**] As best viewed in **FIG. 3B, a** specific example is diagrammed which illustrates how a number of files, their descriptions, and the users that own them might be represented in the directory database **112**. In file directory table **350**, for instance, the HASH ID **354**, the file size **356** and file type **358** are categorized and stored in the central server **110** for each particular file **351***a*-**351***c*. Correspondingly, in the elements table **360**, the HASH ID **364**, the file name **362**, the metadata **366** and the peer Id **368** are categorized and stored as well. Finally, in user table **370**, the Peer ID **374**, the address location **372** and the "online" determination, are categorized and stored on the server.

[**0038**] File **351***a*, for example, refers to a specific file whose HASH code is "A," whose length in "n" bytes, and whose type is "MP3." Two users, "Peer 1" and "Peer 2" each have a copy of this file **351***a*. Peer 1 has named file **351***a* as "Hotel California" in the file name **362** and described it as "Eagles" (in this case the name of the artist that recorded that song) in the metadata **366**. Peer 2, in contrast, has named file **351***a* as "California, Hotel" in the file name **362** and described it simply as "Music File" in the metadata **366**. Peer 1 and Peer 2 are both currently online, so that any other peer wishing to obtain either "Hotel California" or "California, Hotel," both of which ultimately have HASH code "A", will have two choices as to which machine to obtain it from.

[**0039**] Each peer **102***a*-**102***n* in the peer-to-peer network **100** maintains their own local directory of what files (e.g., **451***a*-**451***c*) it has available. In one particular embodiment, this directory may be stored as a local XML file for easy exchange with the server **110** over internet **104**. **FIG. 4A** illustrates the contents of a local directory on a specific peer. The local file table **450** contains a record for each file **451***a*-**451***c*. Each file may be described by its filename **452** (which may be subject to the constraints of the local operating system as to permitted characters, format, and length), a HASH ID **453**, a file size **454**, a file type **457**, file meta data **458**, and a location or folder path **459**. The latter facilitates the peer device to quickly locate the file on the local disk when another peer device requests it.

[**0040**] Other files may be added to the local peer directory whenever the user drags them into one or more specific folders on the local machine. Files may also be deleted from the local directory when they are deleted from those folders or moved out of the specific directories. Alternative implementations may use different business rules and user interface processes to specify when a file becomes available for sharing.

[**0041**] To determine what files are available and where they may be available in the central directory **112** so that they can readily be searched by any peer **102***a*-**102***n*, a directory synchronization process is necessary. Applying this technique, it may be the responsibility of each peer

102*a*-**102***n* in the network **100** to send their list of files they have available, at least periodically, along with any changes to that list as they occur, to the central directory **112**.

[**0042**] Referring now to one specific implementation, **FIG. 4B** shows the directory synchronization process between the local peer directory **450** and the central directory **112** commencing at operation **400**. Initially at operation **402**, the selected local files to be made available for file sharing are identified. The HASH code of each file to be shared (if it has not already been done previously or if a file has changed for any reason) is computed in operation **403**. Once the HASH code for each new or changed file has been computed, the entire local directory **450**, or alternatively only changes and additions to the local directory, are transmitted to the central server **110** in operation **404** via the internet **104**. The central server **110** then proceeds to synchronize each individual file with the central directory **112**. The first central directory operation **405** checks to see if a file with that particular HASH ID already exists in the file directory **310** of data directory structure **300** (**FIG. 3A**). If it doesn't, operation **406** is performed to add the new file to the central directory **112**. In either case, the system now proceeds to operation **407**, which is to check to see if a record exists in the elements table **320** for this particular file and user combination. If it does not, operation **408** is performed to add the filename, meta data, and peer ID to the elements table. If an element already existed for that user ID and file ID in the elements table **320**, that record is checked for any required changes and updated as necessary. In any event, the new data coming from the peer **102***a*-**102***n* takes precedence over corresponding existing data in the central directory **112**.

[**0043**] **FIG. 4C** further illustrates an alternative embodiment process of adding new files into the local directory **470**. This file addition process may be performed whenever a new location (a file, several files, or a folder containing several files) is added to the list of locations available for file sharing or whenever a new file is added to an existing shared folder. Whenever a new file is identified in operation **474**, its HASH code is immediately determined in operation **476**. The local directory **450** is updated in operation **478** with the particular information for that file, including its filename, HASH ID, size, and location on the peer device.

[**0044**] **FIG. 5** illustrates the process of searching for files according to one specific implementation. When a requesting peer1 (**504**) searches for a fall or partial filename or a particular keyword, the peer software sends the search request to the central server **502** in operation (1). The central server **502** processes this request and generates a list of matching files at operation (**3**), returning it to the requesting peer1 (**504**) in operation (**5**). The peer1 (**504**) displays only the relevant information to the user, who then selects one or more files to retrieve in operation (**7**). The requesting peer1 (**504**) then sends the HASH ID of one or more files to be retrieved to the server **502** at operation In operation (**11**), the server identifies zero, or one or more "on-line" locations (addresses) at which the requested file(s) may be found. The list of HASH Ids and matching locations is returned to the requesting peer1 (**504**) in operation (**13**). Finally, in operation (**15**), the requesting peer initiates the procedure to retrieve the selected file(s) from the locations provided by the server **502**, the operations of which are described below.

[**0045**] In one embodiment, the returned list in operation (**13**) may be limited to a maximum number of locations for each file, since in practice a file can usually be reliably retrieved from a relatively small number of locations. In an alternative embodiment, the server **502** may have returned the possible locations of each file with the results of operation (**5**). This would be beneficial by virtue of saving the second query to the server in operation (**9**) but would be expensive since locations would have to be found and transmitted, even for files that are not ultimately desired. In addition, the locations an available file may change in the intervening time between the original search and the selection of particular files to retrieve.

[**0046**] Once a list of locations for a file has been identified, the requesting peer1 (**504**) can choose a location from the list (assuming there is more than one location) using any number of different techniques. The requesting peer1 (**504**) can just pick the first location from the list, pick a location at random, or use some heuristic algorithm to find the "best" location to retrieve from. This may involve "pinging" each location to determine its relative distance on the network **100** from the requesting peer1 (**504**), or requesting a 1024 byte packet from each location to see which one can deliver bytes fastest. Such heuristics may also be useful to see which peers the requesting peer can complete a connection to. It may be impractical for the server **502** to know which peers can communicate efflciently with what other peers at any given time, since the server may not have sufflcient knowledge of the topology of the network **100** or of the trafflc loads that exist on it at any particular time.

[**0047**] In **FIG. 6**A, in accordance with one specific embodiment, once the requesting peer1 (**504**) has chosen a first location to start retrieving the file from (e.g., peer2 (**506**) in this case), the requesting peer1 makes a request at operation (17) to peer2 (**506**) for a file that has the desired HASH ID. Peer2 (**506**) then identifies which file in its local directory corresponds to the desired HASH ID, for example, by performing a lookup or search in its local directory **450** at operation (**19**). If the requesting peer1 (**504**) and peer2 (**506**) can reliably communicate, peer2 transmits the contents of the requested file to the requesting peer1 at operation (**21**). Once the file has been successfully retrieved, the file is stored locally in operation (**23**).

[**0048**] The name and meta data attached to the file at this point will be that which was originally selected in the search results in operation (**5**) of **FIG. 5**. This name and meta data may not necessarily be the same as the name and meta data attached to the file by peer2 (**506**). Finally, in operation (**24**), the requesting peer1 notifies the server **502** that it has a copy of the file, in this way potentially becoming a fulfilling peer for a subsequent request for this same file. Furthermore, this final message from the requesting peer can be used by the server to log successful file transfers, helping operators monitor the efflciency of the network **100**.

[**0049**] In some situations, such as the presence of firewalls, proxy servers, or other network devices, or the fact that the fulfilling peer is no longer online, the first fulfilling peer2 (**506**) may not answer a request for a file. **FIG. 6B** illustrates this situation, in one specific implementation. If the fulfilling peer2 (**506**) does not respond within a nominal timeout interval, the requesting peer1 (**504**) will select the next location Peer3 (**508**) from the list of available locations determined in operation (**13**) of **FIG. 5**, using any one of a number of heuristic algorithms to do so. Assuming in this case that Peerl (**504**) and Peer3 (**508**) can communicate, Peerl will send a request for the file with the specified HASH ID to Peer3 in operation (**27**). Next, Peer3 will find that file on the local peer device in operation (**29**), and will transmit the contents of the requested file to Peer 1 in operation (**31**). Once the file has been successfully retrieved, the file is stored locally in operation (**33**). Finally, in operation (**34**), the requesting peer (**504**) will notify the server **502** that it now has a copy of the file.

[**0050**] According to one specific implementation, the retrieving of a file from one fulfilling peer that is interrupted for any reason may be resumed from another peer that is online and that has that file. **FIG. 6C** illustrates the situation where a file may be partially retrieved from the first fulfilling peer2 (**506**) in the list of locations determined at operation (**13**) of **FIG. 5**. After a time, the first fulfilling Peer2 (**506**) is no longer providing the contents of the file to the requesting peerl (**504**) at operation (**21***a*). The requesting Peerl (**504**) detects a timeout in operation (**35**)**1** and then decides to proceed with retrieving the remainder of the file from a second fulfilling Peer3 (**508**). The requesting peerl (**504**) in this case makes a request in operation (**37**) for the file with HASH ID starting at a position one byte greater than the amount of the file retrieved so far from the next peer in the list of available locations; in this case Peer3 (**508**).

[**0051**] It is crucial that the file contents on the first fulfilling Peer2 (**506**) and the second fulfilling Peer3 (**508**) corresponding to HASH ID be identical in every respect, since otherwise these parts of files may not fit together correctly and result in a damaged or corrupted final file. This is why it is essential to pick a HASH function that will uniquely create a unique HASH code from a file's contents.

[**0052**] Once the second fulfilling Peer3 (**508**) identifies which file in its local directory corresponds to the desired HASH ID at operation (**39**), the new fulfilling Peer3 (**508**) returns the remainder of the file to the requesting peerl (**504**) in operation (**41**). The application in the requesting peerl (**504**) then joins the two chunks of the file together in operation (**43**) and stores the file locally. In operation (**44**), the requesting peerl (**504**) will notify and update the central directory **112** of the server **502** that it now has a copy of the file.

[**0053**] In an alternative embodiment, when the relative ability of multiple peers to deliver files is not known, it can be advantageous to retrieve different parts of a single file from multiple peers **102***a*-**102***n* simultaneously. This may be particularly true if the requesting peer has a faster connection than most of the fulfilling peers since the file can be retrieved faster than any single fulfilling peer can deliver it.

[**0054**] According to one specific implementation shown in **FIG. 7**, the requesting peerl (**504**) requests different "chunks" of the desired file from two different fulfilling peers—(**506**) and (**508**). The partial file requests in operation (**2***a*) and operation (**2***b*) takes the form of the HASH ID of the desired file, the starting position in the file, and the end position in the file. In this example, the requesting peerl (**504**) may request for chunks of size "n" bytes. The chunk size "n" may be statically or dynamically determined based upon parameters such as, size of file to be retrieved and/or number of peers which currently have the file available. The first partial request in operation (**2***a*) may be for the first "n" bytes goes to peer3 (**508**), starting a byte no. **1** and ending at byte no. n. The second partial request in operation (**2***b*) may be for the second "n" bytes goes to peer2 (**506**), starting a byte no. n+1 and ending at byte no. **2***n*. Each fulfilling peer returns the requested part of the file to the requesting peerl (**504**) in operations (**8***a*) and (**8***b*). In operation **43**, the requesting peerl (**504**) reassembles the chunks of the file received from each peer in the right order into the actual file. The requesting peerl (**504**) will then notify and update the central directory **112** of the server **502** that it now has a copy of the file which is not shown.

[**0055**] A file may be retrieved from multiple fulfilling peers in parts or "chunks." According to one specific embodiment, it may be the responsibility of the requesting peer to keep track of what chunks have already been retrieved, what chunks are currently being retrieved, and what chunks remain to be retrieved. **FIG. 8** illustrates a "chunk map"**800** constructed by a "chunk manager" tool or application in which a file has been divided into "m" chunks. Each chunk will typically have the same size, for example "n" bytes, except for the last chunk which may have an odd size since there may be no guarantee that the requested file can be divided into a number of equal sized chunks. According to this specific embodiment, each chunk exists in one of three possible states: AR=Assigned and Retrieved; ANR=Assigned and Not Retrieved; and NAR=Not Assigned and Not Retrieved. The chunk manager uses this state information to determine which chunks are to be retrieve next. The file is known to have been completely retrieved when every chunk is in the "AR" state.

[**0056**] In one specific example, a file whose size is 1.45 MB, for instance, is being requested. If the system has been configured to use a value of n=100 kb, there will be m=15 chunks, each of size 100 kb, except for the 15th chunk, whose size will be 50 kb.

[**0057**] According to one specific implementation, the chunk manager can assign the retrieval of any one chunk of a file to a worker thread (i.e. Peerl-PeerN). Multiple worker threads may be running in parallel, each retrieving a distinct chunk of the file. The chunk manager may employ a variety of techniques to assign chunks to retrieve and peers to retrieve from to different threads. In one embodiment, the chunk manager assigns "p" chunks sequentially to "p" individual threads. Typically, $p \leqq m$, the number of chunks available, although an alternative may be to use more threads than there are chunks and to simply terminate the surplus threads that are not finished when the entire file has been retrieved. The number of threads that can be run in parallel may be constrained by system resources available on the peer device, by operating system constraints, or for any other reason.

[**0058**] **FIG. 9** illustrates one specific embodiment where a chunk manager executive thread at operation (**900**) manages the efforts of multiple worker threads. The executive thread starts with initial parameters at operation (**902**) including a list of peers that have the file with HASH ID that is to be retrieved, as well as the chunk size "n" to be used in retrieving. Alternatively, the chunk manager may compute its own value of n based on the number of peers available and the size of the file.

[0059] The executive thread's first task at operation (904) is to launch a number "p" of worker threads. Each worker thread starts up in operation (932) and waits for an assignment in operation (934).

[0060] The executive thread assigns a chunk to each available worker thread in operation (906). Each unit of work may be characterized by the HASH ID of the file to be retrieved, the peer to retrieve it from, and the start and end positions in the file. Each worker thread accepts a work assignment in operation (936), and then makes a request to the assigned peer for the assigned chunk in operation (938). This request includes the HASH ID of the file and the start and end positions in the file. Meanwhile, the executive thread waits for chunks to be received in operation (908).

[0061] A query is performed at operation (940) about whether or not the desired chunk has been completely received. If "YES", when a worker thread has completely received a chunk, it sends that chunk or preferably, a reference to the location of the chunk in memory or on a storage device, to the chunk manager in operation (942). The chunk manager accepts the chunk at operation (910), and updates its chunk map 800, marking a i the received chunk as having state "AR" in operation (912).

[0062] If the query performed at operation (940) is answered with a "NO", the failure is reported to the executive manager at operation (944). This reported information will include the HASH ID of the file to be retrieved, the peer to retrieve it from, and the start and end positions in the file of the chunk not retrieved.

[0063] At this point, regardless of the query at operation (940), the worker thread that has just finished returns to operation 934 and waits for another assignment. The executive thread examines its updated chunk map in operation 914 to see if there are any unassigned chunks (state "NAR") remaining. If there are, it selects one unassigned chunks in operation (916) and assigns it to a free worker thread in operation (906).

[0064] If there are no unassigned chunks, the executive thread may decide to select an existing assigned, but not yet fully received "ANR" chunk, for reassignment in operation (920). This decision may be made based upon a variety of factors, including the current rate of retrieving of unfinished chunks, the availability of additional worker threads or peers, or the relative retrieve speed of available peers. If an assigned, but unfinished chunk is selected for reassignment, the existing worker thread assigned to that chunk has its assignment terminated and becomes available for reassignment at operation (922). The chunk it was working on is now marked as "NAR" and is ready for reassignment in operation (906).

[0065] In one specific embodiment, a specific peer may be attached to a specific worker thread for the duration of the process. Worker threads that finish sooner may get new work assigned to them, with that new work being targeted to peers that are faster at delivering chunks of the file. Should a peer fail to deliver a chunk in a timely manner, it may be removed from the list of available peers and the thread may request a new peer to interact with. The delivery speed of various peers may vary over time, so that what was a fast peer at the beginning of the process becomes a slow peer towards the end or vice-versa. Since new work tends to go to threads that are finishing their work fastest, the system self-optimizes the retrieve to deliver the file as fast as possible.

[0066] By way of example, in the previously discussed file of FIG. 8 where the 1.45 MB file is divided into m=fifteen (15) chunks of n=100 kb (the $_{15}$th chunk being 50 kb), ten (10) peers are online and have the file with HASH ID available. The executive manager decides to use p=eight (8) worker threads to retrieve the file. If every chunk retrieved is successful and is performed in the same amount of time, each thread will retrieve two 100 kb chunks, except for the 8th thread, which will only retrieve one 100 kb chunk, and for the 7th thread, which will retrieve a 100 kb chunk followed by a 50 kb chunk.

[0067] In practice, some threads will be assigned to unresponsive peers and will fail to retrieve their chunks the first time. The executive thread may mark these peers as unresponsive and assign new peers to the available threads. Furthermore, some peers may be much faster than others at retrieving chunks. In that case, they will become available for retrieving new chunks earlier, the result being that one thread may retrieve five (5) or six (6) chunks while the remainder only retrieve one or two.

[0068] Referring now to FIG. 10, one specific embodiment of a possible process for dealing with peers that fail to respond for chunks of a file is illustrated. The executive thread receives a failure notice from a worker thread in operation (1002). It then sets the measured speed of the corresponding peer to zero (0) in operation (1004). Subsequent peer assignments may use a ranking of measured speeds to pick the fastest available peers rather than the slower, or non-responsive ones. According to a specific embodiment, each peer may be assigned an average or nominal speed prior to the start of file retrieving. The speed of each peer would then be set to the actual speed as chunks are actually delivered. Since delivery speed may vary on a minute-by-minute basis, the most recent measurement of peer speed may be deemed to be authoritative. At operation (1006) the status of identified peers as being non-responsive are updated.

[0069] FIG. 12 illustrates one example of a user interface 1200 on the peer device according to a specific embodiment of the present invention. The user interface, for instance, may be implemented as an application running on the Microsoft Windows operating system.

[0070] In one example, as viewed in FIG. 13, the user interface 1200 includes a search input field 1300 where the user can search for files. In this example the search term "clinton" is entered in field 1300. Files that are found are displayed in a search results list 1301.

[0071] The user can then select one or more files from the list 1301 of search results, as shown in FIG. 14, to retrieve from other peers. In this instance, the mpeg file "clintong.mpeg"1400 is highlighted.

[0072] It will be appreciated that the technique of the present invention provides improved peer-to-peer networking technology for enabling faster and more reliable downloads, using multiple peers in a round-robin or simultaneous retrieving mode, and/or being able to resume failed downloads from different peers. According to a specific embodiment, at least a portion of these features may be implemented by identifying files based on their contents rather than their file names. In this way it is possible to identify and retrieve file content from one or more identical files stored in the peer-to-peer network which have different file names and/or other metadata descriptors.

[0073] According to a specific embodiment, the peer-to-peer network of the present invention includes a central directory, like Napster, but unlike Gnutella which uses a distributed directory. However, it will be appreciated that the technique of the present invention may be applied to both to central directory systems as well as peer-to-peer, distributed directory systems.

[0074] Other Embodiments

[0075] Generally, the peer-to-peer file sharing techniques of the present invention may be implemented on software and/or hardware. For example, they can be implemented in an operating system kernel, in a separate user process, in a library package bound into network applications, on a specially constructed machine, or on a network interface card. In a specific embodiment of this invention, the technique of the present invention is implemented in software such as an operating system or in an application running on an operating system.

[0076] A software or software/hardware hybrid implementation of the peer-to-peer file sharing technique of this invention may be implemented on a general-purpose programmable machine selectively activated or reconfigured by a computer program stored in memory. Such programmable machine may be a network device designed to handle network traffic, such as, for example, a router or a switch. Such network devices may have multiple network interfaces including frame relay and ISDN interfaces, for example. Specific examples of such network devices include routers and switches. For example, the technique of the present invention may be implemented on specially configured routers or servers such as specially configured router models **1600, 2500, 2600, 3600, 4500, 4700, 7200, 7500**, and **12000** available from Cisco Systems, Inc. of San Jose, Calif. A general architecture for some of these machines will appear from the description given below. In an alternative embodiment, the peer-to-peer file sharing technique of this invention may be implemented on a general-purpose network host machine such as a personal computer or workstation. Further, the invention may be at least partially implemented on a card (e.g., an interface card) for a network device or a general-purpose computing device.

[0077] Referring now to **FIG. 11, a** network device **60** suitable for implementing the peer-to-peer file sharing techniques of the present invention includes a master central processing unit (CPU) **62**, interfaces **68**, and a bus **67** (e.g., a PCI bus). When acting under the control of appropriate software or firmware, the CPU **62** may be responsible for implementing specific functions associated with the functions of a desired network device. For example, when configured as a server device, the CPU **62** may be responsible for analyzing packets, encapsulating packets, forwarding packets to appropriate network devices, processing file search requests, maintaining shared file information across the peer-to-peer network, etc. Alternatively, when configured as a peer network device, the CPU **62** may be responsible for initiating file search requests, retrieving file content information from peer devices, performing hash coding operations on selected files, etc. The CPU **62** preferably accomplishes all these functions under the control of software including an operating system (e.g. Windows NT), and any appropriate applications software.

[0078] CPU **62** may include one or more processors **63** such as a processor from the Motorola family of microprocessors or the MIPS family of microprocessors. In an alternative embodiment, processor **63** is specially designed hardware for controlling the operations of network device **60**. In a specific embodiment, a memory **61** (such as non-volatile RAM and/or ROM) also forms part of CPU **62**. However, there are many different ways in which memory could be coupled to the system. Memory block **61** may be used for a variety of purposes such as, for example, caching and/or storing data, programming instructions, etc.

[0079] The interfaces **68** are typically provided as interface cards (sometimes referred to as "line cards"). Generally, they control the sending and receiving of data packets over the network and sometimes support other peripherals used with the network device **60**. Among the interfaces that may be provided are Ethernet interfaces, frame relay interfaces, cable interfaces, DSL interfaces, token ring interfaces, and the like. In addition, various very high-speed interfaces may be provided such as fast Ethernet interfaces, Gigabit Ethernet interfaces, ATM interfaces, HSSI interfaces, POS interfaces, FDDI interfaces and the like. Generally, these interfaces may include ports appropriate for communication with the appropriate media. In some cases, they may also include an independent processor and, in some instances, volatile RAM. The independent processors may control such communications intensive tasks as packet switching, media control and management. By providing separate processors for the communications intensive tasks, these interfaces allow the master microprocessor **62** to efficiently perform routing computations, network diagnostics, security functions, etc.

[0080] Although the system shown in **FIG. 11** illustrates one specific network device of the present invention, it is by no means the only network device architecture on which the present invention can be implemented. For example, an architecture having a single processor that handles communications as well as routing computations, etc. is often used. Further, other types of interfaces and media could also be used with the network device.

[0081] Regardless of network device's configuration, it may employ one or more memories or memory modules (such as, for example, memory block **65**) configured to store data, program instructions for the general-purpose network operations and/or other information relating to the functionality of the peer-to-peer file sharing techniques described herein. The program instructions may control the operation of an operating system and/or one or more applications, for example. The memory or memories may also be configured to include.

[0082] Because such information and program instructions may be employed to implement the systems/methods described herein, the present invention relates to machine readable media that include program instructions, state information, etc. for performing various operations described herein. Examples of machine-readable media include, but are not limited to, magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as CD-ROM disks; magneto-optical media such as floptical disks; and hardware devices that are specially configured to store and perform program instructions, such as read-only memory devices (ROM) and random access memory (RAM). The invention may also be embodied in a carrier wave travelling over an appropriate medium such as airwaves, optical lines, electric lines, etc. Examples of program instructions include both machine code, such as produced by a compiler, and files containing higher level code that may be executed by the computer using an interpreter.

[0083] Additional embodiments of the present invention are described in Appendix C and Appendix D to the present application, each of which is incorporated herein by reference in its entirety for all purposes. Appendix C is entitled, "FLYCODE DATABASE SPECIFICATION", and Appendix D is entitled, "FLYCODE VERSION 2 ARCHITECTURE—SPECIFICATION".

[0084] Although several preferred embodiments of this invention have been described in detail herein with reference to the accompanying drawings, it is to be understood that the invention is not limited to these precise embodiments, and that various changes and modifications may be effected therein by one skilled in the art without departing from the scope of spirit of the invention as defined in the appended claims.

APPENDIX A

The MD5 Message-Digest Algorithm

The MD5 Message-Digest Algorithm

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

Acknowlegements

We would like to thank Don Coppersmith, Burt Kaliski, Ralph Merkle, David Chaum, and Noam Nisan for numerous helpful comments and suggestions.

Table of Contents

1. Executive Summary

This document describes the MD5 message-digest algorithm. The algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA.

The MD5 algorithm is designed to be quite fast on 32-bit machines. In addition, the MD5 algorithm does not require any large substitution tables; the algorithm can be coded quite compactly.

The MD5 algorithm is an extension of the MD4 message-digest algorithm 1,2]. MD5 is slightly slower than MD4, but is more "conservative" in design. MD5 was designed because it was felt that MD4 was perhaps being adopted for use more quickly than justified by the existing critical review; because MD4 was designed to be exceptionally fast, it is "at the edge" in terms of risking successful cryptanalytic attack. MD5 backs off a bit, giving up a little in speed for a much greater likelihood of ultimate security. It incorporates some suggestions made by various reviewers, and contains additional optimizations. The MD5 algorithm is being placed in the public domain for review and possible adoption as a standard.

For OSI-based applications, MD5's object identifier is

```
md5 OBJECT IDENTIFIER ::=
  iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 5}
```

In the X.509 type AlgorithmIdentifier [3], the parameters for MD5 should have type NULL.

2. Terminology and Notation

In this document a "word" is a 32-bit quantity and a "byte" is an eight-bit quantity. A sequence of bits can be interpreted in a natural manner as a sequence of bytes, where each consecutive group of eight bits is interpreted as a byte with the high-order (most significant) bit of each byte listed first. Similarly, a sequence of bytes can be interpreted as a sequence of 32-bit words, where each consecutive group of four bytes is interpreted as a word with the low-order (least significant) byte given first.

Let x_i denote "x sub i". If the subscript is an expression, we surround it in braces, as in x_{i+1}. Similarly, we use ^ for superscripts (exponentiation), so that x^i denotes x to the i-th power.

Let the symbol "+" denote addition of words (i.e., modulo-2^32 addition). Let X <<< s denote the 32-bit value obtained by circularly shifting (rotating) X left by s bit positions. Let not(X) denote the bit-wise complement of X, and let X v Y denote the bit-wise OR of X and Y. Let X xor Y denote the bit-wise XOR of X and Y, and let XY denote the bit-wise AND of X and Y.

3. MD5 Algorithm Description

We begin by supposing that we have a b-bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of eight, and it may be arbitrarily large. We imagine the bits of the message written down as follows:

m_0 m_1 ... m_{b-1}

The following five steps are performed to compute the message digest of the message.

3.1 Step 1. Append Padding Bits

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512.

Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

3.2 Step 2. Append Length

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.)

At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let M[0 ... N-1] denote the words of the resulting message, where N is a multiple of 16.

3.3 Step 3. Initialize MD Buffer

A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

```
word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10
```

3.4 Step 4. Process Message in 16-Word Blocks

We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

```
F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XZ v Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(Z))
```

In each bit position F acts as a conditional: if X then Y else Z. The function F could have been defined using + instead of v since XY and not(X)Z will never have 1's in the same bit position.) It is interesting to note that if the bits of X, Y, and Z are independent and unbiased, the each bit of F(X,Y,Z) will be independent and unbiased.

The functions G, H, and I are similar to the function F, in that they act in "bitwise parallel" to produce their output from the bits of X, Y, and Z, in such a manner that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased. Note that the function H is the bit-wise "xor" or "parity" function of its inputs.

This step uses a 64-element table T[1 ... 64] constructed from the sine function. Let T[i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times abs(sin(i)), where i is in radians. The elements of the table are given in the appendix.

Do the following:

```
/* Process each 16-word block. */
For i = 0 to N/16-1 do

  /* Copy block i into X. */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* of loop on j */

  /* Save A as AA, B as BB, C as CC, and D as DD. */
  AA = A
  BB = B
  CC = C
  DD = D

  /* Round 1. */
  /* Let [abcd k s i] denote the operation
       a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
  [ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
  [ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
  [ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]

  /* Round 2. */
  /* Let [abcd k s i] denote the operation
       a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
  [ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
  [ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
  [ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]

  /* Round 3. */
  /* Let [abcd k s t] denote the operation
       a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
  [ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
  [ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
  [ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]

  /* Round 4. */
  /* Let [abcd k s t] denote the operation
       a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
  [ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
  [ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
  [ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]

  /* Then perform the following additions. (That is increment each
     of the four registers by the value it had before this block
     was started.) */
  A = A + AA
  B = B + BB
  C = C + CC
  D = D + DD

end /* of loop on i */
```

3.5 Step 5. Output

The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

This completes the description of MD5. A reference implementation in C is given in the appendix.

4. Summary

The MD5 message-digest algorithm is simple to implement, and provides a "fingerprint" or message digest of a message of arbitrary length. It is conjectured that the difficulty of coming up with two messages having the same message digest is on the order of 2^64 operations, and that the difficulty of coming up with any message having a given message digest is on the order of 2^128 operations. The MD5 algorithm has been carefully scrutinized for weaknesses. It is, however, a relatively new algorithm and further security analysis is of course justified, as is the case with any new proposal of this sort.

5. Differences Between MD4 and MD5

The following are the differences between MD4 and MD5:

1. A fourth round has been added.

2. Each step now has a unique additive constant.

3. The function g in round 2 was changed from (XY v XZ v YZ) to (XZ v Y not(Z)) to make g less symmetric.

4. Each step now adds in the result of the previous step. This promotes a faster "avalanche effect".

5. The order in which input words are accessed in rounds 2 and 3 is changed, to make these patterns less like each other.

6. The shift amounts in each round have been approximately optimized, to yield a faster "avalanche effect." The shifts in different rounds are distinct.

References

[1] Rivest, R., "The MD4 Message Digest Algorithm", RFC 1320, MIT and RSA Data Security, Inc., April 1992.

[2] Rivest, R., "The MD4 message digest algorithm", in A.J. Menezes and S.A. Vanstone, editors, Advances in Cryptology - CRYPTO '90 Proceedings, pages 303-311, Springer-Verlag, 1991.

[3] CCITT Recommendation X.509 (1988), "The Directory - Authentication Framework."

APPENDIX A - Reference Implementation

This appendix contains the following files taken from RSAREF: A Cryptographic Toolkit for Privacy-Enhanced Mail:

global.h -- global header file

md5.h -- header file for MD5

md5c.c -- source code for MD5

For more information on RSAREF, send email to <rsaref@rsa.com>.

The appendix also includes the following file:

mddriver.c -- test driver for MD2, MD4 and MD5

The driver compiles for MD5 by default but can compile for MD2 or MD4 if the symbol MD is defined on the C compiler command line as 2 or 4.

The implementation is portable and should work on many different plaforms. However, it is not difficult to optimize the implementation on particular platforms, an exercise left to the reader. For example, on "little-endian" platforms where the lowest-addressed byte in a 32-bit word is the least significant and there are no alignment restrictions, the call to Decode in MD5Transform can be replaced with a typecast.

A.1 global.h

```
/* GLOBAL.H - RSAREF types and constants
 */

/* PROTOTYPES should be set to one if and only if the compiler supports
  function argument prototyping.
The following makes PROTOTYPES default to 0 if it has not already
```

```
  been defined with C compiler flags.
 */
#ifndef PROTOTYPES
#define PROTOTYPES 0
#endif

/* POINTER defines a generic pointer type */
typedef unsigned char *POINTER;

/* UINT2 defines a two byte word */
typedef unsigned short int UINT2;

/* UINT4 defines a four byte word */
typedef unsigned long int UINT4;

/* PROTO_LIST is defined depending on how PROTOTYPES is defined above.
If using PROTOTYPES, then PROTO_LIST returns the list, otherwise it
  returns an empty list.
 */
#if PROTOTYPES
#define PROTO_LIST(list) list
#else
#define PROTO_LIST(list) ()
#endif
```

A.2 md5.h

```
/* MD5.H - header file for MD5C.C
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.
```

```
These notices must be retained in any copies of any part of this
documentation and/or software.
 */

/* MD5 context. */
typedef struct {
  UINT4 state[4];                                        /* state (ABCD) */
  UINT4 count[2];        /* number of bits, modulo 2^64 (lsb first) */
  unsigned char buffer[64];                              /* input buffer */
} MD5_CTX;

void MD5Init PROTO_LIST ((MD5_CTX *));
void MD5Update PROTO_LIST
  ((MD5_CTX *, unsigned char *, unsigned int));
void MD5Final PROTO_LIST ((unsigned char [16], MD5_CTX *));

A.3 md5c.c

/* MD5C.C - RSA Data Security, Inc., MD5 message-digest algorithm
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */

#include "global.h"
#include "md5.h"

/* Constants for MD5Transform routine.
 */
```

```
#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21

static void MD5Transform PROTO_LIST ((UINT4 [4], unsigned char [64]));
static void Encode PROTO_LIST
  ((unsigned char *, UINT4 *, unsigned int));
static void Decode PROTO_LIST
  ((UINT4 *, unsigned char *, unsigned int));
static void MD5_memcpy PROTO_LIST ((POINTER, POINTER, unsigned int));
static void MD5_memset PROTO_LIST ((POINTER, int, unsigned int));

static unsigned char PADDING[64] = {
  0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* F, G, H and I are basic MD5 functions.
 */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

/* ROTATE_LEFT rotates x left n bits.
 */
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* FF, GG, HH, and II transformations for rounds 1, 2, 3, and 4.
Rotation is separate from addition to prevent recomputation.
 */
#define FF(a, b, c, d, x, s, ac) { \
 (a) += F ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
  (a) += (b); \
   }
#define GG(a, b, c, d, x, s, ac) { \
  (a) += G ((b), (c), (d)) + (x) + (UINT4)(ac); \
  (a) = ROTATE_LEFT ((a), (s)); \
  (a) += (b); \
   }
#define HH(a, b, c, d, x, s, ac) { \
  (a) += H ((b), (c), (d)) + (x) + (UINT4)(ac); \
  (a) = ROTATE_LEFT ((a), (s)); \
  (a) += (b); \
   }
#define II(a, b, c, d, x, s, ac) { \
  (a) += I ((b), (c), (d)) + (x) + (UINT4)(ac); \
  (a) = ROTATE_LEFT ((a), (s)); \
  (a) += (b); \
   }

/* MD5 initialization. Begins an MD5 operation, writing a new context.
 */
void MD5Init (context)
MD5_CTX *context;                                            /* context */
{
  context->count[0] = context->count[1] = 0;
  /* Load magic initialization constants.
*/
  context->state[0] = 0x67452301;
  context->state[1] = 0xefcdab89;
  context->state[2] = 0x98badcfe;
  context->state[3] = 0x10325476;
}

/* MD5 block update operation. Continues an MD5 message-digest
  operation, processing another message block, and updating the
  context.
 */
void MD5Update (context, input, inputLen)
MD5_CTX *context;                                            /* context */
unsigned char *input;                                    /* input block */
unsigned int inputLen;                         /* length of input block */
{
  unsigned int i, index, partLen;

  /* Compute number of bytes mod 64 */
  index = (unsigned int)((context->count[0] >> 3) & 0x3F);

  /* Update number of bits */
  if ((context->count[0] += ((UINT4)inputLen << 3))
    < ((UINT4)inputLen << 3))
  context->count[1]++;
   context->count[1] += ((UINT4)inputLen >> 29);

   partLen = 64 - index;

   /* Transform as many times as possible.
 */
   if (inputLen >= partLen) {
  MD5_memcpy
    ((POINTER)&context->buffer[index], (POINTER)input, partLen);
  MD5Transform (context->state, context->buffer);

  for (i = partLen; i + 63 < inputLen; i += 64)
    MD5Transform (context->state, &input[i]);

  index = 0;
   }
   else
  i = 0;

   /* Buffer remaining input */
   MD5_memcpy
  ((POINTER)&context->buffer[index], (POINTER)&input[i],
   inputLen-i);
 }

 /* MD5 finalization. Ends an MD5 message-digest operation, writing the
   the message digest and zeroizing the context.
  */
 void MD5Final (digest, context)
 unsigned char digest[16];                          /* message digest */
 MD5_CTX *context;                                        /* context */
 {
   unsigned char bits[8];
   unsigned int index, padLen;

   /* Save number of bits */
   Encode (bits, context->count, 8);

   /* Pad out to 56 mod 64.
 */
   index = (unsigned int)((context->count[0] >> 3) & 0x3f);
   padLen = (index < 56) ? (56 - index) : (120 - index);
   MD5Update (context, PADDING, padLen);

   /* Append length (before padding) */
   MD5Update (context, bits, 8);
  /* Store state in digest */
  Encode (digest, context->state, 16);

  /* Zeroize sensitive information.
*/
  MD5_memset ((POINTER)context, 0, sizeof (*context));
}

/* MD5 basic transformation. Transforms state based on block.
 */
static void MD5Transform (state, block)
UINT4 state[4];
unsigned char block[64];
{
  UINT4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

  Decode (x, block, 64);

  /* Round 1 */
  FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
  FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
  FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
  FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
  FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
  FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
  FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
  FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
  FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
  FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
  FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
  FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
  FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
  FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
  FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
  FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

 /* Round 2 */
  GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
  GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
  GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
  GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
  GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
  GG (d, a, b, c, x[10], S22,  0x2441453); /* 22 */
  GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
  GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
  GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
  GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
  GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */
GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

/* Round 3 */
HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
HH (b, c, d, a, x[ 6], S34,  0x4881d05); /* 44 */
HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

/* Round 4 */
II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */

state[0] += a;
state[1] += b;
state[2] += c;
state[3] += d;

/* Zeroize sensitive information.
```

```
*/
  MD5_memset ((POINTER)x, 0, sizeof (x));
}

/* Encodes input (UINT4) into output (unsigned char). Assumes len is
  a multiple of 4.
 */
static void Encode (output, input, len)
unsigned char *output;
UINT4 *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4) {
 output[j] = (unsigned char)(input[i] & 0xff);
 output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
 output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
 output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
  }
}

/* Decodes input (unsigned char) into output (UINT4). Assumes len is
  a multiple of 4.
 */
static void Decode (output, input, len)
UINT4 *output;
unsigned char *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4)
 output[i] = ((UINT4)input[j]) | (((UINT4)input[j+1]) << 8) |
   (((UINT4)input[j+2]) << 16) | (((UINT4)input[j+3]) << 24);
}

/* Note: Replace "for loop" with standard memcpy if possible.
 */

static void MD5_memcpy (output, input, len)
POINTER output;
POINTER input;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
 output[i] = input[i];
}

/* Note: Replace "for loop" with standard memset if possible.
 */
static void MD5_memset (output, value, len)
POINTER output;
int value;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
 ((char *)output)[i] = (char)value;
}
```

A.4 mddriver.c

```
/* MDDRIVER.C - test driver for MD2, MD4 and MD5
 */

/* Copyright (C) 1990-2, RSA Data Security, Inc. Created 1990. All
rights reserved.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */

/* The following makes MD default to MD5 if it has not already been
  defined with C compiler flags.
 */
#ifndef MD
#define MD MD5
#endif

#include <stdio.h>
#include <time.h>
#include <string.h>
#include "global.h"
#if MD == 2
#include "md2.h"
#endif
#if MD == 4
```

```
#include "md4.h"
#endif
#if MD == 5
#include "md5.h"
#endif

/* Length of test block, number of test blocks.
 */
#define TEST_BLOCK_LEN 1000
#define TEST_BLOCK_COUNT 1000

static void MDString PROTO_LIST ((char *));
static void MDTimeTrial PROTO_LIST ((void));
static void MDTestSuite PROTO_LIST ((void));
static void MDFile PROTO_LIST ((char *));
static void MDFilter PROTO_LIST ((void));
static void MDPrint PROTO_LIST ((unsigned char [16]));

#if MD == 2
#define MD_CTX MD2_CTX
#define MDInit MD2Init
#define MDUpdate MD2Update
#define MDFinal MD2Final
#endif
#if MD == 4
#define MD_CTX MD4_CTX
#define MDInit MD4Init
#define MDUpdate MD4Update
#define MDFinal MD4Final
#endif
#if MD == 5
#define MD_CTX MD5_CTX
#define MDInit MD5Init
#define MDUpdate MD5Update
#define MDFinal MD5Final
#endif

/* Main driver.

Arguments (may be any combination):
  -sstring - digests string
  -t       - runs time trial
  -x       - runs test script
  filename - digests file
  (none)   - digests standard input
 */
int main (argc, argv)
int argc;
```

```
char *argv[];
{
  int i;

  if (argc > 1)
 for (i = 1; i < argc; i++)
   if (argv[i][0] == '-' && argv[i][1] == 's')
     MDString (argv[i] + 2);
   else if (strcmp (argv[i], "-t") == 0)
     MDTimeTrial ();
   else if (strcmp (argv[i], "-x") == 0)
     MDTestSuite ();
   else
     MDFile (argv[i]);
  else
 MDFilter ();

  return (0);
}

/* Digests a string and prints the result.
 */
static void MDString (string)
char *string;
{
  MD_CTX context;
  unsigned char digest[16];
  unsigned int len = strlen (string);

  MDInit (&context);
  MDUpdate (&context, string, len);
  MDFinal (digest, &context);

  printf ("MD%d (\"%s\") = ", MD, string);
  MDPrint (digest);
  printf ("\n");
}

/* Measures the time to digest TEST_BLOCK_COUNT TEST_BLOCK_LEN-byte
  blocks.
 */
static void MDTimeTrial ()
{
  MD_CTX context;
  time_t endTime, startTime;
  unsigned char block[TEST_BLOCK_LEN], digest[16];
  unsigned int i;
  printf
 ("MD%d time trial. Digesting %d %d-byte blocks ...", MD,
  TEST_BLOCK_LEN, TEST_BLOCK_COUNT);

  /* Initialize block */
  for (i = 0; i < TEST_BLOCK_LEN; i++)
 block[i] = (unsigned char)(i & 0xff);

  /* Start timer */
  time (&startTime);

  /* Digest blocks */
  MDInit (&context);
  for (i = 0; i < TEST_BLOCK_COUNT; i++)
 MDUpdate (&context, block, TEST_BLOCK_LEN);
  MDFinal (digest, &context);

  /* Stop timer */
  time (&endTime);

  printf (" done\n");
  printf ("Digest = ");
  MDPrint (digest);
  printf ("\nTime = %ld seconds\n", (long)(endTime-startTime));
  printf
 ("Speed = %ld bytes/second\n",
  (long)TEST_BLOCK_LEN * (long)TEST_BLOCK_COUNT/(endTime-startTime));
}

/* Digests a reference suite of strings and prints the results.
 */
static void MDTestSuite ()
{
  printf ("MD%d test suite:\n", MD);

  MDString ("");
  MDString ("a");
  MDString ("abc");
  MDString ("message digest");
  MDString ("abcdefghijklmnopqrstuvwxyz");
  MDString
 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789");
  MDString
 ("1234567890123456789012345678901234567890\
1234567890123456789012345678901234567890");
}

/* Digests a file and prints the result.
```

```
 */
static void MDFile (filename)
char *filename;
{
  FILE *file;
  MD_CTX context;
  int len;
  unsigned char buffer[1024], digest[16];

  if ((file = fopen (filename, "rb")) == NULL)
 printf ("%s can't be opened\n", filename);

  else {
 MDInit (&context);
 while (len = fread (buffer, 1, 1024, file))
   MDUpdate (&context, buffer, len);
 MDFinal (digest, &context);

 fclose (file);

 printf ("MD%d (%s) = ", MD, filename);
 MDPrint (digest);
 printf ("\n");
  }
}

/* Digests the standard input and prints the result.
 */
static void MDFilter ()
{
  MD_CTX context;
  int len;
  unsigned char buffer[16], digest[16];

  MDInit (&context);
  while (len = fread (buffer, 1, 16, stdin))
 MDUpdate (&context, buffer, len);
  MDFinal (digest, &context);

  MDPrint (digest);
  printf ("\n");
}

/* Prints a message digest in hexadecimal.
 */
static void MDPrint (digest)
unsigned char digest[16];
{
  unsigned int i;

  for (i = 0; i < 16; i++)
 printf ("%02x", digest[i]);
}
```

A.5 Test suite

The MD5 test suite (driver option "-x") should print the following results:

```
MD5 test suite:
MD5 ("") = d41d8cd98f00b204e9800998ecf8427e
MD5 ("a") = 0cc175b9c0f1b6a831c399e269772661
MD5 ("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5 ("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5 ("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b
MD5 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") =
d174ab98d277d9f5a5611c2c9f419d9f
MD5 ("123456789012345678901234567890123456789012345678901234567890123456
78901234567890") = 57edf4a22be3c955ac49da2e2107b67a
```

Security Considerations

The level of security discussed in this memo is considered to be sufficient for implementing very high security hybrid digital-signature schemes based on MD5 and a public-key cryptosystem.

Author's Address

Ronald L. Rivest
Massachusetts Institute of Technology
Laboratory for Computer Science
NE43-324
545 Technology Square
Cambridge, MA 02139-1986

Phone: (617) 253-5880
EMail: rivest@theory.lcs.mit.edu

APPENDIX B

IP Authentication Using Keyed SHA1 with Interleaved Padding (IP-MAC)

IP Authentication using Keyed SHA1 with Interleaved Padding (IP-MAC)

Status of this Memo

This memo defines a Historic Document for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document describes the use of keyed SHA1 with the IP Authentication Header.

Table of Contents

1. Introduction

The Authentication Header (AH) [RFC-1826] provides integrity and authentication for IP datagrams. This specification describes the AH use of keys with the Secure Hash Algorithm (SHA1) [FIPS-180-1]. This SHA1-IP-MAC algorithm uses a leading and trailing key (a variant of the "envelope method"), with alignment padding between both keys and data.

> It should be noted that this document specifies a newer version of SHA than that described in [FIPS-180], which was flawed. The older version is not interoperable with the newer version.

This document assumes that the reader is familiar with the related document "Security Architecture for the Internet Protocol" [RFC-1825], that defines the overall security plan for IP, and provides important background for this specification.

1.1. Keys

The secret authentication key shared between the communicating parties SHOULD be a cryptographically strong random number, not a guessable string of any sort.

The shared key is not constrained by this transform to any particular size. Lengths of 160-bits (20 octets) MUST be supported by the implementation, although any particular key may be shorter. Longer keys are encouraged.

1.2. Data Size

SHA1's 160-bit output is naturally 32-bit aligned. However, many implementations require 64-bit alignment of the following headers.

Therefore, several options are available for data alignment (most preferred to least preferred):

1) only the most significant 128-bits (16 octets) of output are used.

2) an additional 32-bits (4 octets) of padding is added before the SHA1 output.

3) an additional 32-bits (4 octets) of padding is added after the SHA1 output.

4) the SHA1 output is variably bit-positioned within 192-bits (24 octets).

The size and position of the output are negotiated as part of the key management. Padding bits are filled with unspecified implementation dependent (random) values, which are ignored on receipt.

Discussion:

Although truncation of the output for alignment purposes may appear to reduce the effectiveness of the algorithm, some analysts of attack verification suggest that this may instead improve the overall robustness [PO95a].

1.3. Performance

Preliminary results indicate that SHA1 is 62% as fast as MD5, and 80% as fast as DES hashing. That is:

SHA1 < DES < MD5

This appears to be a reasonable performance tradeoff, as SHA1 internal chaining is significantly longer than either DES or MD5:

DES < MD5 < SHA1

Nota Bene:

Suggestions are sought on alternative authentication algorithms that have significantly faster throughput, are not patent-encumbered, and still retain adequate cryptographic strength.

2. Calculation

The 160-bit digest is calculated as described in [FIPS-180-1]. A portable C language implementation of SHA1 is available via FTP from ftp://rand.org/pub/jim/sha.tar.gz.

The form of the authenticated message is:

```
SHA1( key, keyfill, datagram, datafill, key, sha1fill )
```

First, the variable length secret authentication key is filled to the next 512-bit boundary, using the same pad-with-length technique defined for SHA1. The padding technique includes a length that protects arbitrary length keys.

Next, the filled key is concatenated with (immediately followed by) the invariant fields of the entire IP datagram (variant fields are zeroed). This is also filled to the next 512-bit boundary, using the same pad-with-length technique defined for SHA1. The length includes the leading key and data.

Then, the filled data is concatenated with (immediately followed by) the original variable length key again. A trailing pad-with-length to the next 512-bit boundary for the entire message is added by SHA1 itself.

Finally, the 160-bit SHA1 digest is calculated, and the result is inserted into the Authentication Data field.

Discussion:

The leading copy of the key is padded in order to facilitate copying of the key at machine boundaries without requiring re-alignment of the following datagram. Filling to the SHA1 block size also allows the key to be prehashed to avoid the physical copy in some implementations.

The trailing copy of the key is not necessary to protect against appending attacks, as the IP datagram already includes a total length field. It reintroduces mixing of the entire key, providing protection for very long and very short datagrams, and robustness against possible attacks on the IP length field itself.

When the implementation adds the keys and padding in place before and after the IP datagram, care must be taken that the keys and/or padding are not sent over the link by the link driver.

A. Changes

Changes from RFC 1852:

Use of SHA1 term (as always intended).

Added shortened 128-bit output, and clarify output text.

Added tradeoff text.

Changed padding technique to comply with Crypto '95 recommendations.

Updated references.

Updated contacts.

Minor editorial changes.

Security Considerations

Users need to understand that the quality of the security provided by this specification depends completely on the strength of the SHA1 hash function, the correctness of that algorithm's implementation, the security of the key management mechanism and its implementation, the strength of the key, and upon the correctness of the implementations in all of the participating nodes.

The SHA algorithm was originally derived from the MD4 algorithm [RFC-1320]. A flaw was apparently found in the original specification of SHA [FIPS-180], and this document specifies the use of a corrected version [FIPS-180-1].

At the time of writing of this document, there are no known flaws in the SHA1 algorithm. That is, there are no known attacks on SHA1 or any of its components that are better than brute force, and the 160-bit hash size of SHA1 is substantially more resistant to brute force attacks than the 128-bit hash size of MD4 and MD5.

However, as the flaw in the original SHA1 algorithm shows, cryptographers are fallible, and there may be substantial deficiencies yet to be discovered in the algorithm.

Acknowledgements

Some of the text of this specification was derived from work by Randall Atkinson for the SIP, SIPP, and IPv6 Working Groups.

Preliminary performance analysis was provided by Joe Touch.

Padding the leading copy of the key to a hash block boundary for increased performance was originally suggested by William Allen Simpson.

Padding the leading copy of the key to a hash block boundary for increased security was suggested by [KR95]. Including the key length for increased security was suggested by David Wagner.

Padding the datagram to a hash block boundary to avoid (an impractical) key recovery attack was suggested by [PO95b].

References

[FIPS-180] "Secure Hash Standard", Computer Systems Laboratory, National Institute of Standards and Technology, U.S. Department Of Commerce, May 1993.

Also known as: 58 Fed Reg 27712 (1993).

[FIPS-180-1] "Secure Hash Standard", National Institute of Standards and Technology, U.S. Department Of Commerce, April 1995.

Also known as: 59 Fed Reg 35317 (1994).

[KR95] Kaliski, B., and Robshaw, M., "Message authentication with MD5", CryptoBytes (RSA Labs Technical Newsletter), vol.1 no.1, Spring 1995.

[PO95a] Preneel, B., and van Oorshot, P., "MDx-MAC and Building Fast MACs from Hash Functions", Advances in Cryptology -- Crypto '95 Proceedings, Santa Barbara, California, August 1995.

[PO95b] Preneel, B., and van Oorshot, P., "On the Security of Two MAC Algorithms", Presented at the Rump Session of Crypto '95, Santa Barbara, California, August 1995.

[RFC 1320] Rivest, R., "The MD4 Message-Digest Algorithm", RFC 1320, April 1992.

[RFC 1700] Reynolds, J. and J. Postel, "Assigned Numbers", STD 2, RFC 1700, October 1994.

[RFC 1825] Atkinson, R., "Security Architecture for the Internet Protocol", RFC 1825, July 1995.

[RFC 1826] Atkinson, R., "IP Authentication Header", RFC 1826, July 1995.

Contacts

Comments about this document should be discussed on the photuris@adk.gr mailing list.

This document is a submission to the IP Security Working Group of the Internet Engineering Task Force (IETF). The working group can be contacted via the current chairs:

Questions about this document can also be directed to:

Perry Metzger
Piermont Information Systems Inc.
160 Cabrini Blvd., Suite #2
New York, NY 10033

EMail: perry@piermont.com

William Allen Simpson
DayDreamer
Computer Systems Consulting Services
1384 Fontaine
Madison Heights, Michigan 48071

EMail: wsimpson@UMich.edu
wsimpson@GreenDragon.com (preferred)

Editor's Note

This paper was originally submitted May 1, 1996.

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

APPENDIX C

Flycode
Version 2 Architecture-Specification

Flycode
Version 2 Architecture- Specification
(Company Confidential)

Last updated: 12 June, 2001 by Mitra

Purpose

This is the main document for the Version 2 Architecture.

Version Two of Flycode has some similarities to Version 1, but many differences! This is an independent spec, it may (initially) refer to parts of the version 1 spec, but it should not be assumed that anything in the V1 spec still applies unless specifically mentioned here.

This is one of several specification documents which detail the design of the Flycode application. This document will refer to other documents in this specification series and they should be read in conjunction with this document to obtain a complete view of the current design concept.

The reader of this document should also be familiar with the general overview of the Flycode Product.

Nondisclosure

Use of this document and its contents and concepts is governed by a Nondisclosure Agreement signed by anyone authorised to be reading this.

Contents

Back to Flycode Specifications Index

This document is organized as follows.

First there is are two overviews a high-level "100,000 feet" overview, and then a little more detailed "10,000 feet" overview showing the basics of how it works. Read them both first, they should be understandable on their own.

Then there are four interrelated sections.

Functional	Server	Database	Client spec
How different concepts work, across multiple components, with links to the components that implement that functionality.	Module by module through the server.	All changes required to the Database, new tables and changes to existing ones.	Module by Module through the client.
Search Downloading a Folder & Synchronization Metrics and Logging Registration Publishing and Viewing local folders. DRM Auto-Update Thumbnails Unknown Media	registration_after authenticate_form POST authenticate_check authenticate_check_cookie authenticate_cookie comments conditions download file fileinfo folder POST folder folder_hashes folder_xslt	authenticationdomains collections configuration elements files folderelements userfiles users	What is Where; PPH ToolBar, Download Dialog, Launcher, Folder Explorer, Advertising Banner, Search, Local Folder Folder XSLT, Properties, Login,

Copyright and Intellectual Property

This document is the property of Flycode Inc. All software code and concepts designed under this project remain the intellectual property of Flycode

Technical Control

This design and the control of the software specifications are being managed by:
Mitra, Flycode Chief Technology Officer.
<mitra@earth.path.net>

All questions relating to this specification document should go to him.

Then there is a **Open Issues** *section covering outstanding issues:*Not in V2.0, Questions for Tom, Compatibility issues, Email Notes

And a References section with links to some places on the net that might help with implementation.

Terminology: Note that the terms Folder, FileClub and Collection are synonymous.

Next Few things to do here - I'm open to reordering this, so if you have a priority please ask, *note that other items may get slotted into this order.*

- **Tidy up items - do anytime**
  - Go through inprocess.htm - see what to use, what to throw.
  - Include all required bits of V1 spec, commented as unchanged.
- **Required for work in progress now**
  - consider authentication redirection with POST.
- **Required for items needed for May 31st Milestone - basic demo**
  - None
- **Required for items needed for June 15th Milestone**
  - Logging
  - How to catch Windows changes to files / folders (including deleting)
- **Beyond that**
  - DRM / publishing (local Folder/editing)
  - Look at Send2Friend again - **waiting on responses from Tom and Michel**
  - Searching (publishers)
  - Controlling embedding file in page

- Proxy
- Copyright
- Update Distributed Get and incorporate here.
- Chat integration;
- Think about making URLs cacheable e.g. file/1a2b3c
- Integrating ICRA for content ratings;
- About box
- Slideshow

Change Log - use this if you are reading the spec regularly, note that cosmetic etc. changes aren't listed below.

- 12 June
    - Merged V1:Softare Updates into Function:Install/Auto-Update; Server:software_version and Client:Ping . This is not intended to change the functionality, except for removing some V1 parameters no longer required, but it might clarify things.
    - Added Server/Database compatability which is the same as V1.
- 11 June
    - Merged V1:Search_Request into search - no functional changes, but updated spec to include non-documented change in V1 to add <COLLECTION/>.
- 10 June
    - Merged V1:Registration_Request into POST registration_submit - no functional changes
- 8 June
    - Ping added comment that is v2ping for combined V1/V2 server, and merge V1:Ping into it, and defines terms exactly.
    - Merged V1:Upload_Index into POST index - no changes to it.
- 7 June
    - Change to Server:ping and Function:Authenticate to handle an authentication problem.
    - Added comments and locations to replace fileinfo
- 1 June
    - Changed POST folder to return collection id
    - Noted that userids are A-Z0-9 _ and -
- 28 May
    - Updated Server:POST Folder expanded to include full spec from V1 (no changes to spec), and to be specific about how copyright files are handed (copyright handling is not required until after Jun 15th.
    - Client Folder Updater now specifies hash algorithm for collections (same as V1)
    - Added <COPYRIGHT> (post Jun15)
- 24May
    - Updated PPH to use name and dir paramaters,
    - Changed all places that ptptp: URLs are used (find "ptptp:" to see them all)
    - Changed all places that http://server*/file?hash=1a2b URLS are used to add name parameter
    - Changed Folder Downloader to be clear about interface with PPH.
    - Changed Server:File to have name= parameter
- 22May
    - Updated Function:Auto-Update to include installer and make it stand-alone (without ref to V1)

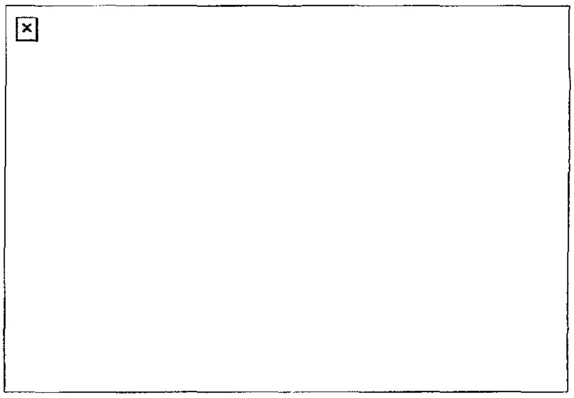
- 18May
    - Clarified Options .
    - Minor correction to server:folder_xslt and added example to Server:URLs to redirect.
- 16 May
    - DRM / Preview: Working on revised preview spec in v2_preview.htm see seperate Change log there.
        - Removed conditions and preview_download and preview_mbox and preview_purchase to V2_Preview, note that comment in V2 about nofile was incorrect, it should always do a redirect.
    - Edited Function:DRM diagram
- 15 May
    - PPH
        - Clarified that must generate events that get to the status line.
        - Should remove illegal characters from filename of temp file.
        - Clarified that on failure no action is required.
    - Adult
        - Not required in folders_containing until later.
- 14th May
    - Updated Download Dialog to show how to select a diferent location for the download.
- 10th May
    - Moved Right-Click and Options to client.
    - Marked Some sections as Jun15 or after Jun 15 to match schedule.
    - Removed Function:URLs section
    - Noted incompatability between V1 and V2 for peer-to-peer component.
    - Noted issue around authenticating POST's
    - Added Logging to PPH and Indexer
- 8th May
    - Clarified authentication domain in V2.0
    - Clarified actions to take for different authentiation commands
    - corrected "&" to "?" in authenticate_check_cookie
- 7th May
    - Registry: Clarified that FlyTemp must be on the same drive as Download, changed the case of FileClubs, added Users key for the hotlist directory and Searches key for the results of searches.
    - Changed a couple of places where the default/example directory for a user's clubs was shown as "My Clubs" instead of "FileClubs".
    - Expanded Right-Click/Add to Hotlist to refer to Directory Structure.
- 2nd May
    - Indexer can keep hierarchy of Folders, related change to Folder Updater.
- 25th April
    - Updated Folder_XSLT, clarified links to use etc. Please read if you are working on the XSLT etc display.
- 24th April
    - Local Folder / Editing: clarified behavior on multiple files, and how to check for a supported file type.
    - DRM - updated and more detail, also involved changing name of server:condition to server:conditions, added preview_mbox, preview_download, preview_purchase,
- 23 rd April
    - Registry - designed, and initial values put into it, any module, refering to an

absolute path now edited to refer to the Registry entry.
- Function:Downloading - corrected to use Folder Downloader
- Added timeline diagram to Function:Authentication
- Deleted Client:status line it will be the standard IE status line, driven by standard PPH responses.
- Client:Search clarified to initialize options from registry (or could be cookie).
- Local Folder & Folder_XSLT: Clarified case of "Remote Folder" link.
- Client:Uploader: Corrected URL, and that it notifies Transfer Manager, not Transfer Panel.
- Deleted OpenIssues:Dependencies, it is no longer relevant.
- Right-Click, clarified Add To Flycode.

- 22nd April
  - Extended Client:Ping to cover going off-line.
  - Clarify some things in Client:Folder_XSLT.
- 21st April
  - Client:Ping, signals Online to other modules, specifically:Folder Updater and Transfer Panel & Indexer (which is also modified to remember changes when offline).
- 20 April - lots of minor changes.
  - Function:URLs: Deleted URL for fetching collection;
  - Server:registration_after redirection for after registering a user.
  - Server:download: correction to go to registration page afterwards.
  - Server:file - show where links to the Client:Launcher.
  - Server:registration - changed to use a "redirect" parameter instead of "hash" to make it compatible with "Server:download"
  - Server:registration_submit - made into a POST, and then changed where it redirects.
  - thumbnail and POST thumbnail clarify where the file stored.
  - Client:PPH - it doesn't return ReportData from HTTPPH to TH, it sends status to Transfer Manager, not Transfer Panel,
  - Corrected Toolbar / Download - it should be active when a Folder is being viewed, not when something is selected.
  - Server:folderscontaining renamed to Server:folders_containing
  - Server:userfolders renamed to user_folders
  - Spell checked it!
  - Fixed all the broken links.
- 19 April
  - Index, Launching, Authentication & Ping section - relates the four concepts
  - Client:Ping, Server:Ping, Server:software_version, Client:authentication; Client:Login; plus related changes to POST index; Server.authenticate; Function:Auto-Update; Systray COM object; deleted Server:login.
- 18 April
  - Transfer Manager - restarts downloads and Transfer Panel; including edits to PPH (checking MD5, making filenames safe); Options; Uploader - set a limit;
- 16 April
  - Clarify a couple of things in Server:stateless
  - Started V2 Design Notes document as a place to put notes that aren't strictly part of the spec, but are worth referring to.
  - Authentication: moved discussion of alternatives to design notes; added Server:Authentication; Database:authenticationdomains and Server: authenticate_form - POST authenticate_check - authenticate_check_cookie - authenticate_cookie

- 15 April
  - More changes to PPH, Uploader, Indexer (changed name from Index Updater), DRM.
  - Folder Downloader and related changes to PPH, Download Dialog; Synchronization;
  - Update: Thumbnails specifying folders;
  - Clarify Systray.
- 12 April
  - Client:What is Where;
  - Function:Serving Files; Listener;Uploader
  - PPH first draft
- 10 April
  - Stateless Server: Including Database:sessions & users & userfiles
  - Authentication;
  - Client:Indexer, Server POST index & changes to Publishing; Folder Data Flow;
- 9 April
  - Client: Systray
- 7 April
  - V1 - V2 Compatibility.
- 6 April
  - Download and Synchronization:Download Dialog: URLs; P2P Fetcher; Server:folder; Synchronization;
  - Reordered Server section, added contents, checked all links
  - Updated Function:Right Click.
  - Client:Folder_XSLT - how things shown.
- 5 April
  - Client: Local Folder
- 4 April
  - Thumbnails: Change to Server:POST Folder; Database:Files; Client Properties; POST Thumbnail;
- 3 April
  - Client - Properties
  - How Indexes get updated: Function Folder Data Flow: Server:folder_hashes; Client:Login; Client:Folder Updater

Overview - 100,000 feet



The Flycode client and server are designed as far as possible to approximate normal browser, web and windows behavior.

- At its simplest, the Client browses web pages and windows folders that are driven from the "folder Server".
- When it selects a file, this goes via Windows standard pluggable protocol mechanisms to the P2P tool, this interacts with the index server and fetches the file from another Peer.
- The fetched file is then launched, again via

Windows mechanisms and will typically launch Windows Media Player.

- Both functions access the database, which unless mentioned here or here is unchanged from V1
- In practice, the Folder and index servers will be the same software on the same machine, but they need not be, and in particular the Folder server could be replaced by anything including a standard web server just serving web pages.
- There are of course, lots of variations and details not mentioned above!

Overview - 10,000 feet

Viewing a Folder

Viewing a Folder occurs in several steps. **Do timeline and examples**

- The user is browsing a normal web page for example from a Flycode customer "Big Big Movies"
- The user comes across a URL to a Folder e.g. http://server.flycode.com:2000/folder?id=123 (or folder?userid=bigbig&name=movies)
- This does a standard query with the browser for this page.
- The server generates a redirection to http://server.flycode.com:2000/folder_xml?id=123
- folder_xml generates the list of items and includes a pointer to http://server.flycode.com:2000/folder_xslt?id=123
- folder_xslt typically redirects to a specific XSLT file, and contains specficiation for viewing folders.
- The XSLT file can contain VB or Javascript or whatever else is needed to cause the browser to render this page properly.
- The page needs to check whether a file is local - via the Indexer's cache-checking method , and check if it requires a licence, and if so whether it already has it via the DRM components.
- The browser now has both the content and style to display the Folder correctly.
- Note that Folders are nestable

Advantages of doing this are that:

- You can have a single stylesheet apply across multiple Folders
- You can change the stylesheet that a Folder uses without affecting its contents
- There can be multiple views of the same Folder by simple extensions. (V2.*)
- The system can support different UI metaphors simultaneously, for example a Folders oriented approach and a more web like approach.

Getting to the file

Within a Folder, the files are just displayed as links. The goal is that when clicked on, it will go to the P2P fetching component.

- If the link is just in a web page - with no guarantee of Flycode being present on the client
    - then it is encoded as http://server.flycode.com:2000/file?hash=1a2b3c&name=coolvideo.avi,
    - The server turns this into a web page with ActiveX Launcher on it that checks for Flycode presence
    - If Flycode is absent, it redirects to http://server.flycode.com:2000/download?redirect="ptptp://server.flycode.com:2000/1a2b3c&name=coolvideo.avi"
        - This page will prompt the user to download Flycode to see the file,
    - Now that Flycode is present (possibly having just been downloaded)
    - It redirects to ptptp://server.flycode.com:2000/1a2b3c&name=coolvideo.avi

- If the link is in a situation where we know that Flycode is present then the URL is ptptp://server.flycode.com:2000/1a2b3c&coolvideo.avi
  - This goes to the PPH module, a Pluggable Protocol Handler

Fetching the File

The Flycode PPH module is a small program run when called by I.E. or something else linking to a ptptp: URL

- It receives a hash, from the caller, via the PPH API.
- It will check with the Indexer's cache, and return if available
- If not, it will access http://server.flycode.com:2000/fileinfo?hash=1a2b3c&location=1
- It will get back XML with the locations.
- It will contact those clients - using the Distributed Get spec - to retrieve the file,
- The file is stored in the cache, and any indexes (including those on the server) updated.

Viewing the File

Once the file has been retrieved, it can - but won't necessarily - be viewed, (for example we could be downloading a FileClub)

The file is passed to the Windows Launcher, typically this means it will end up in Windows Media Player.

Note this does not (yet) handle the case of embedding the file in a UI, if we chose this, then a slight change may be required here instead of launching to WMP.

Function by Function spec across modules and client/server:

Search; Downloading a Folder & Synchronization; Metrics and Logging; Registration; Publishing and Viewing local folders; DRM; Auto-Update; Thumbnails; Unknown Media Types; How things link together; URL, Folder Data Flow; *V1 V2 Compatibility*; Index, Launching, Authentication & Ping; Authentication; Serving Files;

Search

- Ideally is hooked into the Windows Search Panel
- The Search Panel is HTML in the left-hand-panel.
- Goes to the http://server.flycode.com:2000/search?...
- The results in the search are marked the same as for files in a Folder.
- At the top and bottom of the screen (as specified by the XSLT file) are indicators of what the search was for, how many results returned, and Forward & Back buttons.

Downloading a Folder and Synchronization (by Jun15)

- There might be a way to hook this into I.E.'s concept of working off-line, but it does not look like it, and anyway I.E.'s way of doing this is hard to figure out for the user.
- This is launched via right-click on a link to a folder
- or ... when a remote Folder is open, there should be a "Download" button (enabled by the Launcher from the Folder XSLT) in the Toolbar
- The download button should pop up a Download Dialog.
- Which directs the Folder Downloader; to fetch the file.
- Which checks with http://server.flycode.com:2000/folder?id=123&hash=1a2b3c to see if it needs a current

list of the folders content
- and then iterates through the folder instructing PPH to fetch each file,
- and then moves those files to for example "Registry:Downloads/Movies - big big"
- The icon on that folder should be the Flycode icon, plus some indication of whether its going to be synchronized.
- This does NOT change what the client views when going to http://server.flycode.com:2000/folder?id=123 but when viewing items in that folder e.g. by looking at ptptp://server.flycode.com:2000/1a2b3c?name=coolvideo.avi it will find the downloaded file from the Flycode cache.
- The Explorer window should display the Downloads Folder, with Folders - i.e. Downloaded clubs - shown first.
- The Synchronization component will, at startup, and periodically, check for changes and download all new files

Metrics and Logging (15Jun)

Server-side logging remains as current, i.e. logging the Search, and other statistics.

View and Download logging is more complex because of the integration with windows. Any link to ptptp:* ends up at the PPH, and this can be logged. It can be logged with log_view, and if downloaded logged with log_transfer (or transfer_failed).

Files launched directly from Windows won't be logged.

Registration (15Jun)

- Is moved from a dialogue to a web page http://server.flycode.com/registration and registration_submit
- Is visited after installation and after each successful upgrade of the software.

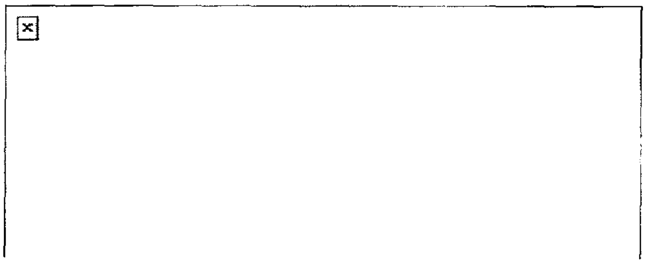
Publishing and Viewing local folders.

- Local Folders are standard Windows Folders with extensions that control how they are viewed and how they react to events like clicking and drag-and-drop.
- The initial folder is under C:/Flycode/jsmith/FileClubs but symbolic links can be created from there to any Folder.
- Files get added and removed using standard windows mechanisms - drag and drop etc.
- Files that are added get meta-data via a Properties dialog which is automatically launched.
- A file index.xml is created in each directory, this contains the properties - or is a copy of properties that are kept in some windows-specific location. It also has an XSLT line.
- Information about his file is sent by the Client: Folder Updater to the server with a POST folder and by the Client: Indexer to the server with a POST index.

DRM

Integrating DRM is a little more complex in V2 than V1 since there are more cases to handle, but the base code remains the same.

The key component is in the XSLT which generates the listing in the folder.



- The page needs to check whether a file is local - via the Indexer and display with or without a check-mark to show the file/Folder is local
- If license=1 then it should use the MLM's methods for checking if a license is present, if a license is required but not present and there is no local copy of the file, then the link should be to http://server.flycode.com/conditions?hash=1a2b&nofile=1

If possible the DRM windows should open in the Explorer Bar, this should work for the Folder Listing -> Condition link, but might not be possible for the cases for the WMP or MLM -> Condition link.

Note the detail on most of these links in the section below on links between components.

To implement this,

MLM: There needs to be an interface to the MLM to decide if there is a licence available on the client. This is used by the Local Folder and by Folder_XSLT for viewing remote folders.

conditions & preview_mbox are changed slightly from V1.

All the JSP pages - such as condition* and accept* remain the same.

asp/offer (or asp/label) will need different DownloadURL and PurchaseURL, see preview_purchase for explanation.

- Research is needed on what parameters the PurchaseURL and DownloadURL get called with, if none i.e.
  if it is just the contents of the appropriate row of zl_url then we will need one row of zl_url per offer with the URL of the conditions for that offer - e.g. http://server.flycode.com:2000/conditions?hash=1a2b, and another for downloading the file with the ptptp URL e.g. ptptp://server.flycode.com:2000/1a3b&name=coolvideo.avi.
- If parameters ARE added, then the Purchase URL can probably be the same for all files, i.e. http://server.flycode.com/preview_purchase, and it can be set at the asp/label level (and automatically inherited by the offer), same for the DownloadURL being http://server.flycode.com/preview_download

Install and Auto-Update (15Jun)

The goal is to inform users of new changes, allowing them to upgrade easily if they wish, and to inform and require upgrades that are neccessary to avoid servers having to be backward compatible.

The client:Ping module requests

http://valerie.flycode.com/intranet/spec/v2_architecture.htm#server_software_version at launch and after each 60 mins (configurable) and uses this to determine if an update is required. If it is, then a seperate installer is downloaded if the software_version indicates a new one required, and then run.

The installer is a small downloadable self-extracting, self-running. (It is kept small by keeping it simple, avoiding fancy graphics, and static linking only those bits of MFC it uses).

The installer, checks a configuration file **(where?)** against the current situation and downloads things that have changed as required. It can also launch other installers such as that for Windows Media Player V7, and Preview's MLM

Thumbnails

- Thumbnails will be stored on the server,
    - This will be a cross-mounted NFS directory. It could in the future also be done using a HTTP server internally accessed, or redirection.
    - Presume they take about 3k bytes each.
    - They will be stored in files called something like .../thumbnails/1/a/2/1a2b3c4d5e.jpg
    - Note that the first three directory levels come from the first three characters of the hash and give us 64^3 = 256k directories.
- Thumbnails of Movies is to be done later, unless an appropriate tool is found, even then we'll probably need to do it Peer to Peer.
- See References/Thumbnails for examples
- Microsoft's Thumbnail control might be useful for displaying, but it looks like it can only handle the original files?
- So another component we need (3rd party, or from Windows) is an image (various formats)->thumbnail (jpg only) converter.
- Thumbnail requests will be a HTTP request so that the browser caches them.
- Thumbnails are not explicitly stored on the client, they rely on the browser cache
- The user interface for viewing thumbnails is defined in Client:Folder_xslt
- When a file is added to a collection, the Properties component is invoked,
- The Properties component notifies the server via both the Folder Updater - which uses POST Folder and the Indexer which uses POST Index
- When a Files record is created - during POST Folder, if it is of type image/*, the server should respond with a Thumbnail required response.
- On receiving this response, the client should do POST Thumbnail?hash=1a2b3c
- The server stores this in its file system
- When a client displays a Folder in Thumbnails mode, then the XSLT (or some other mechanism chosen by Tom) should:
    - fetch the file from http://server.flycode.com:2000/thumbnail?id=1a2b
        - Note that this will automatically check the browser's cache for the thumbnail.
        - This may be redirected to a specific file from whichever server is hosting it.
    - pass it to the Thumbnail control

Unknown Media Types (After 15 Jun)

The User Features specifies auto-installing players - e.g. Flash, QuickTime etc. This depends on the hooks available to catch failure of Windows to launch the file. **Tom to research,** □ if hooks are available then much of the previous specification for this will be incorporated here.

Alternatively, the XSLT could call a function when putting links on a page, and have that check for the existence of the necessary player. It can do this by looking at the registry, and in the absence of an appropriate player, then place a link to a server page that will point at the player.

How things link together

See the DRM diagram for many of these links.

From	To	How
Web page (not necessarily inside Flycode	Folder	http://server.flycode.com:2000/folder?id=123 (or folder? userid=bigbig&name=movies) generates a page that can also be displayed in the Browser, and that will launch Folder extensions etc.
Web Page	File	http://server.flycode.com:2000/file?hash=1a2b3c&name=coolvideo.avi Note that a web page can also include an ActiveX Launcher component and then use links of the form ptptp:....
Folder List	File	ptptp://server.flycode.com:2000/1a2b3c?name=coolvideo.avi is passed to PPH because its a protocol handler for ptptp, and then to Windows launcher which passes it to the Windows Media Player
Folder List	Warning about need for license	http://server.flycode.com:2000/conditions?hash=1a2b&nofile=1 This link is inserted by code in the XSLT for the folder based on the absence of both license and file.
WMP	Condition (and then to license purchase)	If WMP detects that a license is required but not present then it goes to the Purchase URL http://server.flycode.com:2000/conditions?hash=1a2b **(This requires a change to the DRM)**
Windows Launcher	WMP	.asf is specified to open in WMP
PPH	Anywhere	It doesn't - a component does something with a URL that starts "ptptp:", Windows notices that there is a Pluggable Protocol Handler for "ptptp:" and then uses the PPH to get the file or folder, which is then passed along with the action the component has taken. Because this is usually an Open then the file will typically end up in I.E. (if its a Folder) WMP (if it is a video) or somewhere else.
MLM Listing of licenses	WMP (direct or via PPH)	If a user clicks on a license then it either launches the file directly in WMP, or uses the Download URL which will be ptptp://server.flycode.com/1a2b? name=coolvideo.avi and then launches in WMP. **(This requires a change to the DRM)**
MLM Listing	Condition (and then to license purchase)	If the license is expired the MLM Listing directs to the Purchase URL which will be http://server.flycode.com:2000/conditions?hash=1a2b **(This requires a change to the DRM)**
License Purchase	WMP	The license purchase generates a .mbox file - this is returned and launched by the Windows Launcher (unlike V1 which returns and stores this directly) which launches it in the MLM, which fetches and saves the license and then launches the file in WMP.

WMP	Another companies licensing process	A file licensed by someone else will contain their purchase URL, and launch their web page, which should generate an Mbox file which then proceeds as for License Purchase -> WMP.

Folder Data Flow

Meta-data in Folders flows as follows.

- When a file is added to a local folder a Properties dialog is popped up.
- This dialog requests Meta Data and stores it in index.xml in that Folder
  - o it might also store it in some Windows-specific place
- If the client is off-line, then:
  - o the next time the client goes online the Folder Updater module will request http://server.flycode.com:2000/folder_hashes and updates any that have changed with POST Folder
- If the user is online, then the Folder Updater contacts the server with POST Folder
  - o The response to this can request Thumbnails.
- The Indexer keeps the indexes up to date with POST index.
- The folder information is parsed on the server and stored in the database.

V1 - V2 Compatibility

The database changes specified in the Database section are all compatible between V1 and V2, i.e. once the changes are made both servers can coexist on the same data.

In particular, where V1 servers update a record in users or collections they will leave blank the XSLT field, which will be assumed by a V2 server to mean use the default XSLT specified in the configuration parameters url_folder_xslt and url_user_xslt.

Folders containing sub-folders will show up in the folderelements table. V1 clients viewing these sub-folders will not see the extra folders.

There would be one possible problem if a V2 folder with sub-folders was created, and then the same user switched back to V1 and tried to edit the collection.

It is unknown whether V1 and V2 clients can coexist on the same machine, with the same work-area, I suspect NOT and suggest this level of compatibility is a waste of our effort. Instead V2 client should either replace the V1 client, OR be setup with a different data directory.

Note that V1 and V2 clients can NOT interwork at the level of moving files between them. This could be accomplished if required by change "file" to "GetFile"

Index, Launching, Authentication and Ping

It is important to distinguish between three concepts which are separate in V2, but were together in V1.

- Index - the process of telling the server that a particular user and set of files is online at a certain IP address.
- Launching - starting one or more of the components of Flycode
- Authentication - proving that the user is who they say they are.

- Ping - telling the server we are still online.

In a typical scenario,

- At PC startup, the Launcher is run,
- This will start various components including the Indexer module.
- The server checks a cookie, and if absent redirects the client to HTML forms, to authenticate that this is really who the user says they are
- The Ping module connects to the server to tell it that the user is online, and check software version.

Authentication (Dummy by Mar 30, real after Jun 15)

Authentication command by command

Some commands need to be authenticated, in V1 the authentication could be part of the Login command (that was how it was implemented, not how it was specified!). In V2 this is not possible, because different commands may hit different servers, and the server can drop the HTTP connection between commands.

This table categorizes the relationship between server commands and authentication.

Category	**Browser or our code**	**Commands**
Need authentication - because we need one command that can be used to force authentication	Client	ping
Need Authentication because write to database Check the user is the owner of the folder being written	Client	POST folder
Need Authentication because logging with someone's name. Just need authentication to check that we have a name for this user	Client	log_transfer - log_transfer_failed - log_view
Need Authentication because behavior depends on user's identity, just needs to make the userid available to the command.	Browser	conditions, preview_mbox,
	Client	folder_hashes
Don't need authentication in this version, but will need Authentication if reading private resource in later versions.	Browser	folder - folder_xml - folder_xslt - search - thumbnail - user_folders - user_folders_xslt - preview_download -
Don't need Authentication	Client	fileinfo - preview_purchase -
	Browser	folders_containing - logo - search_form - search_xslt
Must not be Authenticated because can be run by non-members.	Browser	download - file - registration - registration_submit - registration_xslt

Part of the authentication process	Client	login

Authentication Alternatives

See V2 Design Notes: Choices for Authentication for a discussion of alternative methods for authentication.

The authentication method chosen is to redirect to a login page when required, request password and then set a cookie.

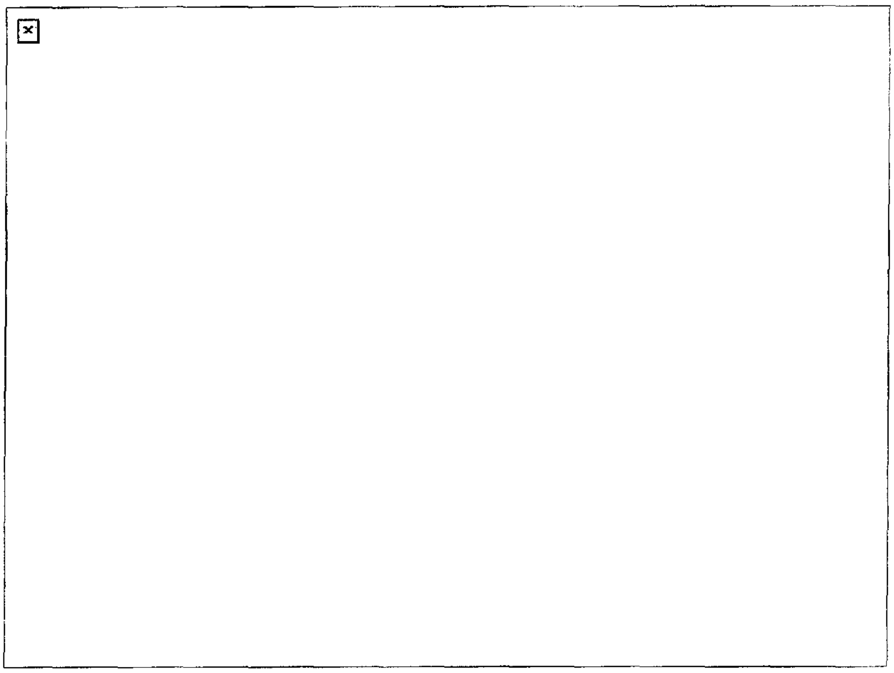
Authentication Requirements

There are a number of requirements that must be met by this spec, which is why it looks a little more complex than at first glance would seem necessary.

- There is a necessity for cross-domain authentication, in both directions e.g.
  - User viewing a web page generated by bigbig.com, needs authentication by Flycode.
  - User viewing a Flycode page, needs authentication by iportal.com
  - User viewing a bigbig web page, needs authentication by iportal.com
- You can't set a cookie in one domain (e.g. Flycode) and read it in another (e.g. BigBig)
- A password must only be entered on a web page belonging to the domain doing the authentication.
- If a user is authenticated for one service, it would be good to be already authenticated when using another service.

Authentication Overview

The following diagram illustrates the time sequence, note that the Auth Server and Web/App Server are the same thing, but do NOT have to be, so there can be no communication between them except via "Redirect", also that the "Server" has no access to the database.



- The server command checks a cookie, and if not present redirects to (a possibly different server or domain)
- authenticate_form : which puts a form together, which when submitted goes to
- POST authenticate_check : which checks it against the database, and if it matches redirects to
- authenticate_check_cookie : which confirms successful cookie writing and redirects (back to the original domain) to
- authenticate_cookie : which sets a domain-specific cookie and redirects back to the server_command.

This sounds complex, but each command is really simple, and it handles all the different cases, and various ways that people might try and hack it.

Note there are two problem cases that need solving - redirecting a POST, and redirection to a form when its a non-interactive command. These need addressing - to do this ...

- The client will try and authenticate ping - there are three possibilities.
  - There is no response - we are offline, so the client will try again later.
  - The server responds with a Redirect, the client passes this to the browser which goes through the authentication process. All otehr client processes continue as if the client was offline until a successful ping is performed.
  - The server responds with the <STATUS> reply as normal, and the client continues with other commands.

Serving Files

- Peers are directed to this Peer by the information contained in a response by the Server to fileinfo?hash=1a2b
- They hit us on port 3000 - or some other port we will define **(Move port to higher number)**
- On this port is the Listener - which accepts the TCP connection, and parses the HTTP command.
- The HTTP command GET File is passed to the Uploader module.
- The Uploader checks with the Indexer to see if we have a file, and if so where.
- The file is fed back to the TCP connection.

Changes to Database required and Migration plan and complete new Database spec

A list of changes required. See also the working document on other database changes not specific to V2.

Please note the V1-V2 Compatibility section, which will need changing if anything in here breaks compatibility.

authenticationdomains - collections - configuration - elements - files - folderelements - userfiles - users

authenticationdomains

Column	Datatype	Description/Comments
rowid	uint	autoincrement
domain	CHAR40	Domain of service - e.g. "bigbig.com"
secret1	CHAR40	Random string shared between bigbig.com and Flycode
cookiegen	CHAR255	URL of cookie gen page at domain e.g. http://www.bigbig.com/cgi-bin/authentication_cookiegen"

collections

Column	Datatype	Description/Comments	Why
xslt	CHAR80	Stylesheet used for this Folder. Note that in the future this might be replaced by one more level of indirection.	See Server: folder_xslt

configuration - extra rows

See the list of URLs that are used for redirections.

Valuename	Example Value	Description

elements

Column	Datatype	Description/Comments	Why
keywords	CHAR255	Keywords - not displayed, but searched on	

files

Column	Datatype	Description/Comments	Why
thumb	~~HASH~~ CHAR1	"1" if there is a thumbnail present.	

folderelements

This table contains folders that are elements of another Folder.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
collectionid	uint	rowid in collection table for where the element appears
name	CHAR60	name of folder as it appears in this folder
time_added	datetime	time at which the Folder element was added
linkto	uint	rowid of collection table for Folder being linked in.
description	CHAR255	file club owner's description of Folder
collections_private	CHAR1	1 if the collection is private, and therefore the elements should not be searched.

sessions

Column	Datatype	Description/Comments
rowid	uint	autoincrement
userid	USERID	user's login id (note that there may be multiple records with a single userid) *consistency should check this matches a row in "users"*
time_created	datetime	Unix timestamp when user logged in.
location	IPPORT	IP Number and port expressed in hex
authenticate	CHAR255	Cached Authentication response (note that this is used only for V1 clients)
port	uint	port to contact the client on
indexhash	HASH	hash supplied with Upload_Index
pingtime	DATETIME	time of last ping

userfiles

Column	Datatype	Description/Comments	Why
~~userid~~	~~CHAR40~~		
sessionid	uint	rowid from sessions table.	

users

Column	Datatype	Description/Comments	Why
xslt	CHAR80	XSLT file to use for user, may be NULL	
~~location~~	~~IPPORT~~	~~IP Number and port expressed in hex~~	

~~authenticate~~	~~CHAR150~~	~~Cached Authentication response~~	
~~port~~	~~uint~~	~~port to contact the client on~~	
~~thread~~	~~uint~~	~~thread number (or other identifier) on that server.~~	
~~time_left~~	~~datetime~~	~~Unix timestamp of when disconnected, set to 0 during session or after cleanup.~~	
userid		Restricted to A-Z0-9_ and -	

Server:

Note that all these commands are additions to the V1 spec, the V1 spec commands will not be required once V2 is operational, but several of the commands here are defined in terms of the V1 spec.

registration_after - authenticate_form - POST authenticate_check - authenticate_check_cookie - authenticate_cookie - comments - conditions - download - file - ~~fileinfo~~ - folder - POST folder - folder_hashes - folder_xml - folder_xslt - folders_containing - POST index - locations - log_transfer - log_transfer_failed - log_view - logo - ping- preview_mbox - preview_download - preview_purchase - registration - POST registration_submit - registration_xslt - search - search_form - search_xslt - software_version - thumbnail - POST thumbnail - user_folders - user_folders_xslt - v2ping - Periodic Actions - Stateless - Authentication - <COPYRIGHT> - Server/Database compatability

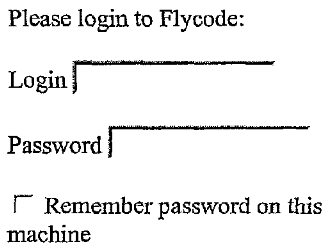
authenticate_form?domain=xyz.com&redirect=http://www.xyz.com/aaa/bbb

Timeline needed

Generate an authentication form for logging in to the Flycode. Include hidden fields for domain and redirect and fields for userid and password and to remember the password

- If error="xxx" is present then display this error. (For example "password didn't match")
- Else if the cookie Authenticate_login is present, then redirect to authenticate_check_cookie? domain=xyz.com&redirect=http://www.xyz.com/aaa/bb

Form when submitted goes to POST authenticate_check.



Please login to Flycode:

Login

Password

□ Remember password on this machine

POST authenticate_check

This should if possible be https, Assume it receives fields "userid=jsmith", "password=xyzzy", "remember=n" from the form, and "domain=xyz.com" and "redirect=http://www.xyz.com/aaa/bbb" from hidden fields.

It checks the userid and password fields against the database.

- If they match then:
  - o set cookie "Authenticate_login=jsmith:1a2b3c" where 1a2b3c=MD5(jsmith ":" password ":" secret) and secret is a secret specific to authenticate_check and authenticate_cookie,
    - If remember=1, then the cookie should be set not to expire, otherwise it should be a session cookie.
  - o redirect to: authenticate_check_cookie?

domain=xyz.com&redirect=http://www.xyz.com/aaa/bbb&remember=n

- else redirect to authentication_form?
domain=xyz.com&redirect=http://www.xyz.com/aaa/bbb&error=Password%20does%20not%20match

authenticate_check_cookie? domain=xyz.com&redirect=http://www.xyz.com/aaa/bbb&remember=n

This has two functions - either check an existing cookie passed to the authenticate_form page, or check that the authenticate_check successfully stored a cookie on the client.

- Read and parse the cookie "Authenticate_login=jsmith:1a2b3c"
- if there is no cookie: redirect to authenticate_form?
domain=xyz.com&redirect=http://www.xyz.com/aaa/bbb&error="You must enable cookies to login to Flycode"
- elsif 1a2b3c=md5(jsmith ":" password ":" secret)
- then redirect to http://xyz.com/authenticate_cookie?
redirect=http://www.xyz.com/aaa/bbb&userid=jsmith&ok=5f6g&remember=n where:
    - http://xyz.com/authentication_cookie came from the cookiegen field of the authentcationdomains table and 5f6g = MD5(jsmith ":" secret)
    - and secret is a secret known only to xyz.com and Flycode and comes from the authenticationdomains table for this domain.
- else redirect to authenticate_form?domain=xyz.com&redirect=http://www.xyz.com/aaa/bbb to rerequest a password.

authenticate_cookie? redirect=http://www.xyz.com/aaa/bbb&userid=jsmith&ok=5f6g&remember=n

This page runs in the domain where the service required exists, which might not be the same as the authentication server.

Check the ok=5f6g field, where 5f6g = MD5(jsmith ":" secret) and secret is the secret shared with Flycode in the authenticationdomains table for this domain.

- If it matches
- then Set a cookie "Authenticate_Flycode=jsmith:566g" where 5f6g is the ok; redirect to the redirect URL.
    - If remember=1 then set to not expire, otherwise make it a session cookie.
- Else log an error, delete the cookie, and redirect to http://www.xyz.com/aaa/bbb (which will redirect on to authenticate_form)

comments?hash=1a2b3c&start=10

Returns a list of comments for a hash from the comments table. This a subset of the V1:GetFileInfo command.

At the moment this is not used in the User Interface for V2 and is not required.

If any comments are found the server should return a 200 status and an XML file, otherwise it should return a 204 and no data.

If the result_set includes comments and the comment_start paramater is present, it specifies where in the list of comments to start sending results.

Comments are presented according to an algorithm that might be changed later:

- First comments from the collection (as specified in the call) are listed in reverse DATETIME order (newest first)
- Then comments not in any collection in DATETIME order
- Then comments from other collections. in DATETIME order
- Only the first 10 are returned.

```
Content-Type: text/xml
Content-Length: 123

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Flycode SYSTEM "http://www.flycode.com/support/flycode.dtd">
<COMMENTS>
<COMMENT userid="jsmith" datetime="98765432" text="I  like this shot, taken at sunset"/>
<COMMENT userid="mpatel" datetime="98765432" text="I  wish he'd learn to focus"/>
</COMMENTS>
```

Comparisom with V1:GetFileInfo: This just looks for comments, and so doesn't have to do any joins, it omits a couple of layers of wrapper XML from the result.

conditions?hash=1a2b3c&nofile=1

See V2_Preview.htm: conditions;

download?redirect=ptptp://server.flycode.com:2000/1a2b?name%3dcoolvideo.jpg&referrer=http://....

This page prompts the user to download Flycode, it should carry the redirect through any subsequent screens, and then redirect to it, via the registration page by going to http://server.flycode.com:2000/registration?redirect=ptptp://server.flycode.com:2000/1a2b?name%3dcoolvideo.jpg after downloading.

file?hash=1a2b3c&name=coolvideo.avi

This needs to direct the client towards getting the file. But, at this point the server does not know if the client has any Flycode components installed.

So this generates an HTML file as described in Client:Folder_XSLT

~~fileinfo?hash=1a2b3c&location=1&comments=1~~

~~This is equivalent to the existing GetFileInfo, returning a set of LOCATION tags, and Comments depending on the other parameters~~

folder?id=123 or ?userid=bigbig&name=movies [&hash=1a2b3c]

This has to generate a page that the client can display.

In the future, if we support multiple ways of representing Folders then it might follows the chain Folders -> styles, and uses the parameters from the styles to generate the page, but for now it just does a redirect to:

http://server.flycode.com/folder_xml?id=123

If called with the userid and name parameters it should look these up and generate the redirect with the id=123 parameter.

If the hash parameter is appended then the server should check the collections.hash field, and if the hash matches should return a result-code of 204 and no data, which the client can interpret as meaning that there is no change.

POST folder?id=123

The server should receive a POST contain an XML file with information about each file in the collection.

- The server uses id=123 to locate the folder by its rowid
  - if no "id" is specified then this is a new collection, except that if it has the same name and owner as a previous collection then it should replace that collection. (This would be the case if the index.xml file got trashed).
- The server should check the hash and oldhash attributes.
  - If the client has included a "hash" attribute of just "" then the Collection has been deleted. Delete from the Collection table and all the relevant elements from "elements" table.
  - If the Client has included a "oldhash" attribute in the COLLECTION element, then this should be checked against the "hash" field of the "collection" table.
    - If it doesn't match, the server should respond with an HTTP 404 error, and the client should re-send a complete list with no "oldhash" value.
  - If no "oldhash" attribute was supplied then this is a replacement, the server should query the collelements table, delete any files that are no longer in the collection. Then, add any new ones - and set their timestamp to now, and update the information (but not the timestamp) on any files that remain in the collection.
- If a matching "oldhash" attribute was supplied, then this is an incremental update, then for each file supplied:
  - If there is a oldhash but hash="" then it is a delete, delete the element.
  - If there is a oldhash and hash value its a change, update the information, but not the timestamp. Note that oldhash=hash is valid and means that the description was changed but the file not changed.
  - If there is a hash but no oldhash then add the file and set the timestamp
- The Server should then set the hash field in the collections table to the "hash" value supplied.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Flycode SYSTEM "http://www.flycode.com/support/flycode.dtd">
<COLLECTION id="123" name="India" description="My photos of India" private="0" oldhash="1a2b3c4d"
hash="5e6f7g8h9i0j1k"
   keywords="india,elephants,delhi">
<FILE hash="1a2b3c" type="image/gif" adult="0" description="Taj Mahal" creator="J.Smith"
file_size="23023" />
<FILE hash="4d5e6f" type="video/avi" adult="0" description="Elephants at the Taj" creator="F.Smith"
file_size="1234567" oldhash="12345abcde"/>
<LINKFOLDER id="345" name="Bruce's Videos" description="Bruce has good taste in videos" adult="0">
</COLLECTION>
```

Note that the userid of the folder MUST match the authentication to allow this.

If the server has to create any records in the files table, then it should check for Thumbnails that are not present by checking that files.thumb=1.

If it is a new collection, then a 200 return code is sent back along with a content.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Flycode SYSTEM "http://www.flycode.com/support/flycode.dtd">
<COLLECTION id="123"/>
```

If there are any thumbnails missing then a 200 return code is sent back along with a content.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Flycode SYSTEM "http://www.flycode.com/support/flycode.dtd">
<THUMBNAILSREQUEST>
<FILE hash="1a2b3c"/>
</THUMBNAILSREQUEST>
```

If both thumbnails and collection id are missing then these can be grouped and returned with a 200 return code..

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Flycode SYSTEM "http://www.flycode.com/support/flycode.dtd">
<GROUP>
<COLLECTION id="123"/>
<THUMBNAILSREQUEST>
<FILE hash="1a2b3c"/>
</THUMBNAILSREQUEST>
</GROUP>
```

If they are all present, then a 204 return code is returned, and no content.

Note that for ease of coding, it is totally valid to enclose just the single (<THUMBNAIL> or <COLLETION> in <GROUP>) or to return empty content with 200 and empty GROUP

The collections:size field should be maintained by this operation, i.e. incremented for every element added, and decremented for every element removed.

If the message includes any files with copyright >1 - other than deleting them from a collection the files should not be added to the table or into the MD5, but the operation should still succeed. Files with copyright = 1 may be added to a collection. For any files that are not added, a <COPYRIGHT/> message should be generated and returned with a return code of 200 (note this doesn't have to happen till after Jun 15).

Otherwise, the server responds with a success code of 204.

If there is an exception or invalid XML then return failure code of 400.

Comparison with V1:Upload_Collection

- The server uses the id to locate the collection rather than userid and name of collection.
- Only one folder should be uploaded at a time (unlike V1 where multiple collections could be in each command).

- A <COLLECTION> element previously only contained <FILE> elements, it may now contain links to FOLDERS.
- The COLLECTION may specify keyboards separate by commas
- COPYRIGHT problems are returned inside the 200 error code (which may or may not be the same as V1).

folder_hashes

Return a list of hashes for the Collections belong to the user, See Function: Folder Data Flow and Client:Folder Uploader for how this is used.

```
<COLLECTIONS>
<COLLECTION name="India" description="My photos of India" private="0" hash="1a2b3c4d5e6f7g8h9i0j1k"/>
<COLLECTION name="Pets" description="My furry critters" private="0" hash="1a2b3c4d5e6f7g8h9i0j1k"/>
</COLLECTIONS>
```

Comparison with V1:<SUMMARY>

- this is the same, except that it should have a <COLLECTIONS> outside object rather than <SUMMARY>.

folder_xml?id=123

This has to generate a page that the client can display.

A typical page might look like

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="http://server.flycode.com/folder_xslt?id=123"?>
<folder>...</folder>
```

Why mix data and rendering? Note that the XSLT is embedded here because it is not currently possible to separate them out and generate a higher-level file that allows generation and reuse of XML which does not refer to the stylesheet.

What about CSS? This would be an alternative, CSS has the disadvantage of not being able to do a transformation, it can only render elements of the XML.

folders_containing?hash=1a2b [&adult=0]

Equivalent to a search, but just retrieve folders containing this file.

```
SELECT collections.* FROM collections,elements WHERE elements.hash=1a2b &
elements.hash=collections.hash
```

The adult flag does not need to be supported until later.

POST index?oldhash=1a2b&hash=3c4d

The server should receive a POST containing an XML file of the hash value for each file available from the user's machine.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Flycode SYSTEM "http://www.flycode.com/support/flycode.dtd">
<FILE_INDEX>
<FILE hash="1234567890123456789012"/>
<FILE hash="5678901212345678901234"/>
</FILEINDEX>
```

The server should check for a record in the sessions table for this IP number and port. (It is important to test both IP number AND port, to distinguish clients connecting through the same HTTP Proxy server). If there is no session record that matches, then create one.

If oldhash is not "" then it is compared against the sessions.indexhash, if it does not match then a 404 error is returned, no changes made to userfiles, and the client will do a full upload.

If oldhash is "" then this is a full update, if there was an old record, then all files for this session should be removed from userfiles,

If the oldhash matches sessions.indexhash then this is an incremental update,

In the case of either oldhash="" or oldhash matches sessions.indexhash then each hashe should be added to userfiles with the session number.

If there are any files in the list with copyright >1 then a <COPYRIGHT....> message should also be returned, files with a copyright >0 should not be added to the database, the operation should still complete successfully.

A success code of 204 should be returned, unless there is a <COPYRIGHT> message returned in which case it is a success code of 200.

Comparisom with V1:Upload_Index

- Works almost the same except for the hashes:
- Note that I believe the V1:Upload_Index does NOT work according to the V1 spec, and does incremental indexes currently.

locations?hash=1a2b3c

This is equivalent to the existing GetFileInfo, returning a set of LOCATION. This a subset of the V1:GetFileInfo command.

The server should look up the hash in the "userfiles" table, If the hash is not found, then a 204 status is returned and no locations. If the hash is found, then the server should return a 200 status, and an XML file.

The server returns Location information neccessary for the client to select a source, and do a GET file. Note that this is separated from the search results so that it can be done as late as possible, to maximize the chance that the chosen location is still available. The Server should look in the "userfiles" table for this hash and join against the sessions table. If there are no locations online then an empty <LOCATIONS> element is returned, with no <LOCATION> *elements.*

```
Content-Type: text/xml
Content-Length: 123
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Flycode SYSTEM "http://www.flycode.com/support/flycode.dtd">
<LOCATIONS>
<LOCATION ip="abcd1234" port="3300">
<LOCATION ip="1234abcd" port="3300">
</LOCATIONS
```

Comparisom with V1:GetFileInfo: This just looks for location and so is a simpler SQL than in V1, not requiring joins against Files or Elements tables, it omits a couple of layers of wrapper XML from the result.

log_transfer?hash=1a2b3c&ip=1.2.3.4&filesize=12345
log_view?....
log_transfer_failed? . (all 15Jun)

All work the same way as currently, except change the case/underlines of the command to be consistent.

ping or v2ping

Ping is used to tell the server that the client is still online, update the ping time in the sessions record, and return a <STATUS> message as for V1:Ping (note this is lower case "ping", compared to V1 upper case "Ping")

Note that ping is authenticated, this allows it to be used as a command to force authentication to happen. See *Function:Authentication*.

The client should send one of these commands every 5 minutes (configurable) and I suggest the server times out anything it hasn't heard from in 10 minutes, and this should be configurable.

There will be somewhere on the V2 client which should give an indication of how many files are online. I suggest something like.

1234 collections containing 123456 files, 75% of which are online, with the count being updated periodically (15 mins)

The server should retrieve COUNT Collections and COUNT FILES and COUNT userfiles. (See note of variance)

This <STATUS> element sets variables that can be used anywhere in the client UI. In this case it will look like.

For efficiency, the server should cache this result and only check every few minutes (Currently it does every 2 mins).it then works out the rate of adding and send ..

```
<STATUS countcollections="1234" deltacollections="12" countuserfiles="12345657" deltauserfiles="123"
   countusersonline="1234" deltausersonline="-5" sumsizeuserfiles="123456" deltasizeuserfiles="123" />
```

The meaning of these is...

Value	Meaning	pseudo-SQL
countusersonline	Number of people online	COUNT sessions

countuserfiles	Total number of files online, counting each duplicates	COUNT userfiles
countcollections	Total number of collections on or offline	COUNT collections
sumsizeuserfiles	Size of files online, counting each duplicates	SUM(files.filesize) FROM userfiles LEFT JOIN files ON files.hash = userfiles.hash

Delta is so-much per minute,

Difference with V1: countusersonline is just COUNT sessions instead of COUNT users WHERE online >0

Spec variances: For the combined server this is v2ping instead.

preview_download
preview_mbox
preview_purchase

See V2_Preview.htm: preview_download; preview_mbox; preview_purchase;

registration?redirect=ptptp://server.flycode.com:2000/1a2b?name%3dcoolvideo.avi

Generates a HTML form for registration, do this by generating an XML form, so that we can brand it in the future.

The <USER> tag should be filled in if the command is authenticated, if not authenticated it should be left blank.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="http://server.flycode.com/registration_xslt"?>
<USER userid="jsmith" email="jsmith@nowhere.com" notice="12">
<REDIRECT file="ptptp://server.flycode.com:2000/1a2b?name=coolvideo.avi">
</USER>
```

The HTML generated by the XSLT should contain Javascript or VB to check the values before submitting.

The HTML form should then be submitted to POST registration_submit.

POST registration_submit

When Flycode is first installed, the new member undertakes a registration process. The is handled by the following message:

The server receives a Simple POST:

```
POST /registration_request
Host: server.flycode.com
Accept-Language: en, fr;q=0.5
Content-Length: 54
Content-Type: application/x-www-form-urlencoded
```

```
userid=jsmith&password=a1b2c3d4&email=jsmith@nowhere.come&notice=12&connection_speed=56k
&redirect=http://server.flycode.com:2000/registration_after
```

The server should look up the userid in the "users" table.

The password is created by the client from the password entered by the user, MD5(userid ':flycode.com:' password). This means that the plain-text password is never sent across the net which is important because its probably used by the user on other web sites. This is the only time that the hashed value is sent over the net, and ideally this call should be hidden in the future by HTTPS.

If the userid is not in the table, then the server should add the record, and add the password hash, and redirects to the redirect supplied, or if that fails to http://server.flycode.com:2000/registration_after

If the userid is in the table, AND this message is authenticated to come from that user, then the registration information and hashed password is updated, and a status code of 204 returned and it redirects to the redirect supplied, or if that fails to http://server.flycode.com:2000/registration_after . Note that it is not possible for this request to change the userid.

If the userid is in the table, but this message is not authenticated from that user, then a status code of 400 should be returned with a string "NAME_FAIL". Note this can occur in two cases, either an unathenticated attempt to register a user with an already existing name. Or an unathenticated attempt to change a registration. (its not clear if this can ever happen in V2)

If the data doesn't pass basic tests then it generates an XML document with errors specified, e.g.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="http://server.flycode.com/registration_xslt"?>
<USER userid="jsmith" email="jsmith.nowhere.com" notice="12">
<ERROR tag="email" text="Not a valid email address">
</USER>
```

Comparisom with V1:Registration_Request: Error handling is different. Instead of a 204 return code on successful new or updated registration, then , there is a redirection to the "redirect" parameter supplied.

search?search_start=0&title=Taj&type=image&online=1&adult=1&keyword=

The server receives a GET containing paramaters for the search,

```
GET /search?search_start=0&title=Taj&type=image&online=1&adult=1&keyword=
Host: server.flycode.com
Accept-Lanuage: en, fr;q=0.5
```

The server should search the "elements" table, the exact match between the query and the search is to be defined later, for now, a title of "Taj" should be matched against "Taj" appearing anywhere in the title field.

Any file with copyright >0 should not be returned.

If type is specified, one or multiple times,then only files with that type or type-prefix are returned, type=club is used for file clubs. For example "type=image" would mean just to return images, "type=club&type=video" would return videos and clubs but not images. If type is not specified then everything is returned.

If online=1 is specified then only files which are online are returned.

If adult=1 is specified then only files with adult <=1 should be returned (i.e. those not specified as adult).

The search_start identified the number (starting with 0) of the first record to send in the result. Since the client is likely to send subsequent requests for the remaining pages of results, the server should cache the results of the search. Note that SEARCH_COUNT in the result is therefore one more than the index of the last search result. The number of results returned in each response should be a configurable parameter.

The server should construct an XML document to return of the form, and return it with a success status code of 200.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Flycode SYSTEM "http://www.flycode.com/support/flycode.dtd">
<?xml-stylesheet type="text/xsl" href="http://server.flycode.com/search_xslt"?>
<SEARCH_RESULTS search_start="0" search_end="50" search_count="75">
<COLLECTION private="0" userid="jsmith" filesize="10" name="India movies" description="Some pics"
hash="1a2b3c"/>
<FILE hash="4d5e6f" name="taj.gif" type="image/gif" adult="0" creator="jsmith"
  description="Taj Mahal" file_size="23023"/ >
<FILE hash="7y8u9i" name="elephants.avi" type="video/avi" adult="0" creator="mpatel"
description="Elephants at the Taj" file_size="1234567"/>
</SEARCH_RESULTS>
```

If there are no results to the Search, then an empty SEARCH_RESULTS is returned, with search_start = search_end = search_count = 0.

Comparisom with V1:Search_Request: Adds an XSL line to the results.

software_version

Request the current software version, which should be returned in the same format as during a V1:Login

The server extracts AVLCLVER, MINCLVER, SETUPURL and SETUPVER values from the database configuration table, and returns:

```
<SOFTWAREVERSION setupversion="20" setupurl="http://www.flycode.com/bAb00/flycodesetup.exe"
available="35" required="35"/>
```

Other paramaters can be put in here, but ONLY if the spec is changed first.

Comparisom to V1:SoftwareVersion: The V1 function had evolved beyond the spec! V2 no longer needs to return the V1 parameters: serverversion, dbversion, uploadport, uploadserver.

thumbnail?hash=1a2b

Redirect to the thumbnail file. See Function:thumbnail for where it goes.

POST thumbnail

POST Thumbnail is used to upload a thumbnail, it is a standard HTTP POST with a file attached, it should contain a parameter hash=1a2b specifying the hash of the file that this is a thumbnail of.

- If the files.thumb field for this hash=0
- then the thumbnail should be stored (See Function:thumbnail for where it goes) and the files.thumb field set to 1, and a return code of 204 sent.
- Else return 404 Thumbnail already exists.

user_folders?userid=jsmith

Displays a page relevant to the user, specifically a list of all folders belonging to that user. This is basically equivalent to V1:Get_User_Collections.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="http://server.flycode.com/user_folders_xslt?userid=jsmith"?>
<COLLECTIONS>
....
</COLLECTIONS>
```

URL's that redirect via the configuration table

Each of these URLs reads a parameter from the configuration table, and redirects to it, if a query is specified, then the value from the configuration table is only returned if the Query returns blank or null.

Why redirect? Because although its marginally less efficient the first time, the file can be cached at the browser, significantly speeding up viewing of the same page again, or of other pages that use the same file.

For example the server command /folder_xslt looks up the configuration paramater url_folder_xslt and finds http://www.flycode.com/v20status/folder.xslt and sends a redirection to it.

URL	**Query**	**Configuration parameter**	**Value, for first release <v20> = "http://www.flycode.com/v20static"**	**What, and any requirements.**
logo		url_logo	<v20>/userstart.htm	When user clicks on Logo button in the toolbar.
search_form		url_search_form	<v20>/search_form.htm	When user clicks on Search button in the toolbar, contains form for searching. See Client:Search
search_xslt		url_search_xslt	<v20>/search.xstl	For presenting results of Search.

user_folders_xslt?userid=jsmith	SELECT xslt FROM user WHERE userid="jsmith"	url_user_folders_xslt	<v20>/userfiles.xslt	For presenting list of user's folders, could be customized by user with logo etc.
registration_xslt		url_registration_xslt	<v20>/registration.xslt	The stylesheet for the Registration form. It should contain a submit button,
folder_xslt?id=12	SELECT xslt FROM collections WHERE rowid="12"	url_folder_xslt	<v20>/folder.xslt	**Future:** the style field might point at a URL that generates the stylesheet dynamically, or be replaced by one more level of indirection.
registration_after?redirect=xxx	(looks forpage specified in "redirect")	url_registration_after	<v20>/registration_after.htm	Start page the user goes to after a new or updated registration if they are not registering as part of clicking on a link to a file.

Periodic Actions

To be completed -

- There may be more changes here, but the presumption is that all periodic actions of V1 remain the same.
- Users should be cleaned up when two pings have been missed, so make this 10 minutes
    - At cleanup of a user, when the files are deleted from userfiles, it should delete the sessions.indexhash
-

Server going Stateless

The V1 server does not appear to retain state (outside of the database) between commands, but it does use the presence or absence of a TCP/IP session as an indicator of whether a user is logged on, and records files by user.

The new server needs to be Stateless, so that commands can happen on separate servers, at the same time. We do this with a "sessions" table that keeps state between TCP/IP sessions from the same client.

The following changes to the V1 server should allow it to support both V1 and V2 clients. It is hard to give an exact specification to the changes because it is not clear exactly how the V1 server works, it does NOT match the V1 spec.

- Make changes to the users table, these have implications to other code.
- Review sql.ini to check for accesses to the fields now moved to the sessions table.
- V1:Login needs to update the sessions table instead of the users table.
- V1:Disconnect needs to update the sessions table instead of the users table, and remove all userfiles that match that session.
- Periodic cleanup will need modifying to delete from the sessions table, and userfiles for that session.
- V1:Upload_Index needs to update the userfiles.sessionid field, not the userfiles.userid field, it can figure out which session record from the IP and userid, but could more easily cache it on the server (after a login) since a V1 client will maintain a single TCP/IP session.
- There should not be a need to change the authentication process for V1, but this needs verification.
- Receipt of a V1:Ping should update sessions.pingtime
- V1 needs to NOT make changes when the TCP/IP session closes, it needs to rely on either a disconnect, or the absence of a Ping to know that a session has ended.

Authentication

When a web page or server command e.g. xyz.com/aaa/bbb (xyz.com could, but need not be flycode.com) needs authentication, it checks the cookie "Authenticate_Flycode=jsmith:1a2b3c", there are three possibilities:

1. 1a2b3c = MD5(jsmith : secret) and secret is specific to that service (for example to Flycode servers), in which case it is authenticated, check that jsmith has permission (normally either all users have permission - e.g. to read something or permission is restricted to the owner), and then carry on with the requested action.
2. If the cookie exists but does not match THEN log a possible hack, delete the cookie and treat as for the next case.
3. If there is no cookie, then Redirect to: http://server.flycode.com:2000/authenticate_form?domain=xyz.com&redirect=http://www.xyz.com/aaa/bbb
    1. Where domain refers to a row in the authenticationdomains table (in V2.0 this can be hard-coded as flycode.com)
    2. and http://xyz.com/aaa/bbb is the page we just came from.

For example .....to authenticate a command http://www.flycode.com:80/log_transfer?x=a&y=b

Then if it arrives without a cookie, then redirect to:

http://server.flycode.com:2000/authenticate_form?
domain=flycode.com&redirect=http://www.flycode.com:80/log_transfer?x%3da%26y%3db

Note that this won't work for redirecting a "POST" (e.g. POST Folder), this needs fixing, short-term workaround is to make sure that first authentication is a GET - and Ping is to be used for this.

<COPYRIGHT .../> (after Jun 15)

When the Server sees that a file, specified in POST Folder or in POST_Index or several other places, is copyright or illegal, it should pass this message back to the client as part of the response. (Note that the name can be omitted if not known to the server, for example in a POST Index

```
<COPYRIGHT/>
  <FILE hash="1a2b3c" name="aaa.jpg" reason="1">
</COPYRIGHT>
```

The client should display this in a dialogue box prompting for deletion

Server <-> Database Compatability

The table "configuration" will contain two rows.

Valuename	Example Value	Description
DBVERSION	1	Version number of database, increment each time a field is added/deleted to ANY table
SWMINVER	220	Minimum version number of server that this database is compatable with. Note that the server version number and minimum version of database are compiled into the server

The server software will have compiled in "server_version_number" and "database_min_version". Note these are compiled into the server, not in the config.ini file in order to minimize the likelihood of the values being just copied over during installation.

Whenever the server opens a new connection to the database it should read the DBVERSION and SWMINVER values and if DBVERSION <= database_min_version or SWMINVER >= sever_version_number then the server should print a message on the console, and that thread should exit. Note that the server does not need to exit since it might in the future have access to multiple databases.

This handles the following cases.

	Requires change in other component	Does not require change
Database upgrade	SWMINVER is increased, Server will connect and find SWMINVER > server_version_number and disconnects	SWMINVER is not changed, server finds DBVERSION has changed, but its still >= database_min_version
Server upgrade	database_min_version increased, server finds DBVERSION < database_min_version and disconnects	database_min_version not changed, server notices change in server_version_number but its still >= SWMINVER.

Client spec: including Windows integration, components and interfaces

Client Modules: What is Where; PPH; ToolBar, Download Dialog, Launcher, Folder Explorer, Advertising Banner, Search, Local Folder, Folder XSLT, Properties, Login, Folder Updater, Indexer, Synchronization; Folder Downloader; Systray; Listener; Uploader; Transfer Manager; Transfer Panel; Authentication; Ping; Login; Registry; Directory Structure; Right Click; Options;

What is available where

The modules described below are available in different places.

Where	What
COM module loaded by the Launcher at Startup	Indexer, Listener, PPH; Synchronization; Systray; Transfer Manager; Ping;
Not clear - Tom to recommend	ToolBar, Download Dialog, Launcher, Folder Explorer, Advertising Banner, Status Bar, Local Folder, Folder XSLT, Properties, Folder Updater,

PPH

This module is used to fetch files, this is called with URLs of the form "ptptp://server.flycode.com:2000/1a2b3c?name=coolvideo.mpg&dir=My%20Videos%20-%20-jsmith" where:

- server.flycode.com:2000 specifies who to ask for File location information.
- 1a2b3c is the hash of the file
- coolvideo.mpg is the name to use for the file after downloading and is required
- dir is where to put the file relative to the download directory (if absent, use the download directory). Note that this cannot be an absolute location, but can be anywhere on the disk if and only if there is an appropriate shortcut in the downloads directory.

It should be implemented as a "Pluggable Protocol Handler", which means that any application that refers to something via ptptp:something will end up doing the expected thing with the resulting file.

I've been going through the PPH documents, it actually looks totally reasonable, the overview.asp is a great resource and seems to lay out step by step how we get called by I.E., except that there are tons of options.

Basically I see the basic structure as this .....

(Abbreviations: PPH or APP is Asynchronous Pluggable Protocol Handler; TM is the Transfer Manager, TP is Transfer Panel, TH is "Transaction Handler" which is the thing which invoked the PPH - typically IE, HTTPPH is the HTTP Protocol Handler module fetching from the peer)

- IE instantiates a PPH for each download - this is detailed in overview.asp (all URLs are below).
- Somehow it has to get a pointer to the TM. I don't see how it can do this since it looks like a fresh IClassFactory is created each time, but I'm not sure I'm understanding this correctly.
- When Start (say with ptptp://server.flycode.com:2000/1a2b3c&name=coolvideo.avi) the PPH should ....
    - check the GetBindInfo call for BINDF flags that make sense. Initially it should only implement the flags actually encountered, and throw an Exception for others, see test cases below.
    - check with the Indexer component for 1a2b3c and if found it has to return this.
    - Otherwise it looks for FlyTemp/1a2b3c and 1a2b3c.info to restart a download
        - 1a2b3c.info should be maintained during the download to enable as easy a retry as possible, restarts should work as present. (I can spell this out if you don't have the email on the Transfer Panel)
        - The filename needs to convert \ / and : to something safe - use the same algorithm for this as in V1.
    - Otherwise it does a HTTP query to http://server.flycode.com:2000/fileinfo?hash=1a2b3c&location=1, and parses the result.
    - It takes the result, and iterates through the locations (note this is the non-distributed get version)
    - For each location e.g. 12.34.56.78 it creates a HTTP protocol handler (HTTPPH) (presumably the same interface as this PPH) for http://12.34.56.78:3000/file/1a2b3c
        - The call needs to set BINDF_NOWRITECACHE (via supplying a

        IInternetBindInfo::GetBindInfo interface for the HTTPPH to call back) to avoid double caching.
        - It needs to implement a IInternetBindInfo::GetBindString interface, that gets called from the HTTPPH to get specific info it needs, this call can be passed straight on to the TH except that if it requests the BINDSTRING_URL then it should return the http: URL rather than pass the call on to the TH which would return a ptptp: URL.
    - It calls this HTTP protocol handler with "IInternetProtocol::Read"
    - As the data is returned it is:
        - put into FlyTemp/1a2b3c
            - *Remove any illegal characters from this hash [/:\]*
        - The file 1a2b3c.info is updated to show how much has been read.
    - The IInternetProtocol::Read calls to the PPH from the TH are responded to from this data.
        - Note that there must be decoupling between the PPH->HTTPPH Reads and the TH->PPH reads so that HTTP retries and eventually distributed get can happen.
    - On completion it should
        - Check the MD5 of the downloaded file, if it doesn't match restart the download (this shouldn't happen)
        - move the file to Registry:Downloads/coolvideo.avi
            - If a file is downloaded with a matching name to one already downloaded, then the name should be changed automatically, append "_6G" where the 6G is the last two characters of the hash.
        - And notify the index handler. **(To be specified)**
        - And log the transfer by a HTTP query to log_transfer.
    - Note that on failure no action is required, the file can be left in FlyTemp for a possible restart.
- There are a number of places where progress reports are needed.
    - The PPH should report to the TH on data as specified in overview.asp using IInternetProtocolSink::ReportData, IInternetProtocolSink::ReportProgress and IInternetProtocolSink::ReportResult.
        - Note - these events must lead to sensible reporting in the status line of IE.
    - The ReportProgress and ReportResult queries should also go to the TM in some TBD form which can use it to update its display when open.
    - But .... the PPH should ignore ReportData calls from the HTTPPH as they refer to a percentage of this particular HTTP read, not a percentage of the file.
    - The PPH could just pass on most (or possibly all) ReportProgress calls from the HTTPPH
    -

Here are some notes on useful pages and on some of the options .... they might not make sense as they are here, but it would be useful for you to have printouts for all of the URLs below when we talk so that we can both refer to them.

- overview.asp READ THIS, it has URLs for other commands only some of which are below.
    - IInternetProtocolRoot/Start.asp
        - IInternetBindInfo/IInternetBindInfo.asp
            - IInternetBindInfo/GetBindInfo.asp info from the TH to the PPH,
                - BINDF.asp its unclear how many of these we need to implement, the overview says to recognize BINDF_NOUI and BINDF_SILENTOPERATION, but we work silently anyway
                - BINDINFO.asp - may not need anything, except that it might need to call for BINDSTRING_POST_COOKIE
                    - BINDVERB.asp - which should always be GET
            - IInternetBindInfo/GetBindString.asp - callback from PPH to TH, will need to support some of these calls from HTTP module we pass it on to,

                - BINDSTRING.asp - Pass all these on from HTTP to TH except for BINDSTRING_URL which should return the HTTP URL.
        - PI_FLAGS.asp
    - IInternetProtocolInfo/ParseUrl.asp
        - PARSEACTION.asp
            - CANONICALIZE, PATH_FROM_URL and URL_FROM_PATH: Return the same URL
            - FRIENDLY: return the name followed by the description if we know it (from GetFileInfo)
            - SECURITY_URL, SCHEMA, SITE, DOMAIN, LOCATION, SECURITY_DOMAIN: I don't know what these need to do, probably the same as HTTP does.
            - ROOTDOCUMENT: Probably http://www.flycode.com would do, I think this is used for figuring out the BASE?
            - DOCUMENT, ANCHOR, ENCODE, DECODE, ESCAPE, UNESCAPE: should all do the same thing as HTTP does.
            - MIME: Should return the value got from GetFIleInfo
            - SERVER: Probably "server.flycode.com:2000"
    - IInternetProtocolSink/ReportProgress.asp
        - BINDSTATUS.asp - make lots of sense to report as many of these as we see (except those that ReportProgress says we shouldnt). Note that overview.asp says that we have to report BINDSTATUS_MIMETYPEAVAILABLE which we can get from the GetFileInfo call.
- Test cases to see what flags / options we have to support.
    - Entered on Address line;
    - Embedded in <A HREF="ptptp:...>;
    - Embedded in <IMG SRC="ptptp:...">;
    - Right-click that image, open in New Window
    - Right-click save to disk
    - From bookmark.
    - All of the above when IE is running off-line.
    - Reload of any of the above, including the off-line ones.

The PPH needs to be able to receive controlling events from the Transfer Panel.

- Delete - Stop the download, remove the temp files, indicate to caller.
- Pause - Temporarily stop the download, leave the temp files, indicate to caller
- Retry - Depends on the state, it should: stop any currently happening download; if it was paused, try the same Peer again, otherwise, move on to the next peer if there is one, and rerequest FileInfo if there is not; continue downloading from the byte-count it got to.

ToolBar

The toolbar sits above the browser window, and is a standard IE toolbar, i.e. it should appear in the IE->View->Toolbars menu and behave just as you would expect a toolbar to behave.

It is started by the Launcher.

The technology to use for the ToolBar is **to be determined by Tom,** it will be whatever integrates easiest as a IE Toolbar.

Image to come later

The toolbar has the following buttons, in left to right order.

Button	**Active When**	**Window navigated**	**To**
Flycode Logo	Always	Main	http://valerie.flycode.com/intranet/spec/v2_architecture.htm#server_redirecturls
Search	Always	Explorer	Search
Transfers	Always	Separate ?	Launch Transfer Panel.
Download	Folder being viewed	Separate	Download Dialog
Send2Friend	File or Folder selected	TBD	
Buddies, MyProfile, Recommend, Report Infringing	V2.1		

Download Dialog

Launched from the Download button on the Toolbar when a Folder is viewed. Passed the id, name and userid of the folder.

It should offer a choice of where to download Folders to, defaulting to [HKEY_CURRENT_USER\Software\Flycode]\Downloads\Funny Videos - Mitra

Also asks □ Keep Synchronized; if Checked ....

The Download Dialog directs the Folder Downloader to fetch the folder and put the files into the place selected above. If this place is NOT the default, then a shortcut should be placed in the default location so that the Synchronization component can walk the tree of downloaded folders.

Launcher

This component ensures that Flycode is running, it needs to be embedable in a Web page, for example as an

ActiveX, or some VB or Javascript code. It also needs to be launchable at startup, or the PPH **(No something else - e.g. TP)** could be launched directly instead.

The first task of the launcher is to figure out if Flycode is installed, if it is, then it can selectively launch components. The parameters that control which components to launch are to be specified when it is clear what technology to use for the Launcher.

It needs a parameter "redirect" which specifies a URL to launch on success.

On failure it should always redirect as below:

What items are launched in which situations.	**Systray COM Object**	**Folder Explorer**	**Toolbar**	**Toolbar - Download**	**Synchronization**	**Advertising**		**URL on success** ... means "server.flycode.com:2000"	**URL for downloading** ... means "server.flycode.com:2000"
Situation	↓	↓	↓	↓	↓	↓			
Web page generated by http://.../file?hash=1a2b&name=aaa.avi	Y	Y	Y	N	Y	Y		ptptp://.../1a2b?name=aaa.avi	http://.../download?redirect="ptptp://.../1a2b?name%3daaa.avi"
Web page including a ptptp link	N	N	N	N	N	N		none	http://.../download
From Folder XSLT	Y	Y	Y	Y	Y	Y			http://.../download?redirect="http://.../folder?id=123"
When called at startup	Y	N	N	N	Y	N		None	N/A (it couldn't be launched at startup if it wasn't on the client already).

Explorer Bar

This is displayed in the standard explorer bar location, subject to usual IE behavior (e.g. appearing in IE->View->ExplorerBar)

It can be launched by the Launcher.

It needs to show a hierarchical table of folders, see Directory Structure for how and where this maps to the disk.

	On Click Main window to	**Notes**
Flycode		Typically C:/Flycode/jsmith

□ Downloads	Registry:Downloads	
□ Funny Movies - bigbig	http://.../collection?id=123	Note that this refreshes the list if we are online.
□ Users		
□ bigbig	http://..../user_folders?user=bigbig	
□ Funny Movies	http://.../collection?id=123	
□ Searches		
□ Funny	C:/Flycode/jsmith/Searches/Funny.xml	
□ Funny Movies - bigbig	http://.../collection?id=123	
□ My Folders	C:/Flycode/jsmith/Fileclubs	
□ HomeSweetHome	C:/Flycode/jsmith/FileClubs/HomeSweetHome	

The technology is probably a HTML page with an embedded OCX, but is **to be determined by Tom**

Advertising Banner (after Jun 15)

This is displayed in between the main panel and the status line. I'm not sure if there is standard IE behavior that this should match..

It can be launched by the Launcher.

It should be HTML, behavior to be determined later.□

Search Panel

- If possible this hooks into the standard Windows Search Pane, so that Flycode is one of the things you get to search. (It seems to be different in Win98, maybe can't hook into all Win9*)
  - Tom says: Explorer "band objects" can implement a search handler (or other custom panels) that plug into the left-hand side (and bottom) of File Explorer. We should be able to derive from default search band and customize.
- The Search panel is an HTML form. (Unless this is precluded by the requirements to hook into Windows Search Panel)
  - It is generated by http://server.flycode.com:2000/search_form
- It has fields for the different Search Options: Adult / Undeclared / Family Safe, Video, Image, Folder, Online
  - These options should be initialized from the Registry (or as a cookie if that is difficult). Before submitting the form it should set these options back in the Registry to initialize the page next time.
- There should be a field to enter the search query.
- V2.1 could add other fields such as Size,
- History of search is provided via the Windows Search Panel ?
- The search navigates the main window to http://server.flycode.com:2000/search?<**same arguments as V1:Search** >□ **except**
  - online=1 specifies to just search files which are online

Local Folder

The local Folder viewer refers to the extensions to the Windows Shell to display our own LOCAL folders.

- This hooks into Windows through two basic mechanisms.
    - o When a local shared folder is opened, it uses Windows Folder Extensions to open a certain view of the club along with appropriate toolbars.
    - o As files are dragged and dropped into the Folder, or deleted from it, the events are trapped and acted on as below.
    - o How exactly this works is for **Tom to specify.**□
- The initial hook is created by creating a folder at installation - Registry:FileClubs, this is marked as a Flycode folder so that it uses the extensions.
- The Folder must be able to switch between Web and Columns views just as for Windows Folders. The Web View should be the same as a user will see - i.e. using the XML & XSLT file.

There are a series of other points where it ties into windows. **The details have to come from Tom.**

For each of the items that can be listed there will need to be hooks to show the correct Flycode Icon instead of the Windows default.

Item	**Icon**	**Display in List**	**On Click** (See Function:Right Click for right click options.
Local File	□ □	Use Properties and/or Meta-data from index.xml, if meta-data indicates needs a license, then use DRM functions to check if need license Can show thumbnails via windows standard mechanisms.	Standard Windows Launch (see DRM diagram for how this can end up somewhere else entirely).
Local Folder	□	Show Collection icon, use Meta-data from that collection's index.xml to show size	Open that folder in Local Collection Viewer (standard windows open)
Remote file	□ □	Not possible	N/A
Remote Folder (stored as a shortcut)	□	Show Collection icon, if available (from Meta-data or because have Synchronized this folder) show collection's size.	Standard IE open of the URL - http://server.flycode.com:2000/folder?id=123

Editing

When a file is dragged into the Folder, the action depends on what is being moved, and whether the user chooses Copy (or Move) or Create Shortcut, in all cases:

- if it is adding a File, the client should check that the File is of a supported type, by checking the registry for a mime type of image/* or video/* and if not should give an error message. If this is a add of multiple files then there should only be ONE error message.
- After successful copy or shortcut creation, of a single file, the client should pop up the Properties dialog box. This should be initialized from the properties of the File or Shortcut being copied.
- After successful copy or move or shortcut creation of multiple files, then do not pop up the Properties dialog

box

- Creating a New Folder is just creating a Windows Folder, setting it to use the extensions, and opening the Properties dialog to collect the Meta-data.

Table of behavior on Dragging, or Cutting and Pasting something into a Local Folder.

What	**Copy or Move**	**Create Shortcut or copy an existing Windows or Internet shortcut**
Local File	Copy or Move the File to the directory	Create a Windows shortcut, No difference in how a Shortcut or the original file is sent to the server.
Local Folder	Copy or Move the Folder and its contents to the directory, take normal windows precautions to prevent recursion etc.	Create a Windows shortcut, No difference in how a Shortcut or the original file is sent to the server
Remote File (e.g. dragging out of a Remote Folder or a Downloaded Folder or the Downloads Folder)	Fetch file if required, then Copy or Move the File to the directory.	Create an Internet shortcut to ptptp://server.flycode.com:2000/1a2b3c?name=coolvideo.avi
Remote Folder or Downloaded Folder	Create an Internet shortcut to http://server.flycode.com:2000/folder?id=123	

Technology: Tom says: Have code examples for catching events on both right and left click drag-n-drop. See COleDataObject.

Need: Link to DRM and encryption for publishing

Folder XSLT

The XSLT is crucial to how this works, it has to generate the Folder listing, its possible that it will require other functionality - e.g. CSS to make it work properly.

The functionality of this file is described throughout this document, but some extra points are: **(gather notes from rest of doc here)**

- Whenever a userid is displayed - e.g. the owner of a Folder, it should be a link.
  - If the display is similar to V1, i.e. that all columns for the file have to be one link, then this can be a right-click item. "Display User's Folders"
  - The link goes to http://server.flycode.com:2000/user_folders?userid=jsmith
- There needs to be code in this that will ensure that the Flycode components are present, and then direct the PPH to get the file for example it might contain an <OBJECT tag> that contained as an ActiveX component,the Launcher; which will ensure toolbars etc. are open. There might be an even simpler way of doing this by loading a 0-pixel image from "ptptp:" and then redirecting as for the launcher depending on whether this succeeds? **(Check with Tom)**
- Note that the Folder XSLT does NOT include the Toolbar or Explorer or Advertising Banner, which are launched by the Launcher as specified above.
- The middle panel should be set up to do previews, and hold meta data. Including Description, Size, Esc, and Comments - the comments will require a separate query to http://server.flycode.com:2000/fileinfo?hash=1a2b&comments=1. How to exactly to drive this middle panel is something to be figured out during

development and separate server queries may be required here. For example, it might be that this panel needs HTML generating, or XML/XSLT or something like that.

- Right Click menus will be added to all items
-

Item	Display in List			On Click (see also Function:Right_Click)
Local File or Local Folder	N/A			N/A
Remote File	Use Meta-data from index.xml, call Indexer to see if have file, if licence=1, indicating needs a license, then use DRM functions to check if need license, icon and link depend on these two questions.			
	Have File	**Licence**	**Icon**	
	Yes	Required, not present	**need icons**	http://server.flycode.com:2000/conditions?id=1a2b
	No	Required, not present	**need icons**	http://server.flycode.com:2000/conditions?id=1a2b&nofile=1
	Yes	Required, present	**need icons**	ptptp://server.flycode.com:2000/1a2b&name=aaa.avi
	No	Required, present	**need icons**	ptptp://server.flycode.com:2000/1a2b&name=aaa.avi
	Yes	Not Required	**need icons**	ptptp://server.flycode.com:2000/1a2b&name=aaa.avi
	No	Not Required	**need icons**	ptptp://server.flycode.com:2000/1a2b&name=aaa.avi
	If showing thumbnails show http://server.flycode.com:2000/thumbnail?id=1a2b			
Remote Folder	Show Collection icon,□ use Meta-data to show size			Standard IE open of the URL - http://server.flycode.com:2000/folder?id=123

Properties

The Properties dialog pops up when either: A file is published (e.g. by dragging into a Local Folder), or a Folder or File is Right-Clicked

If the Properties are for a Remote Folder or a File in a Remote Folder, then the Properties dialog is Read-Only.

The Property Dialog should ask for information for:

- For a File, these properties are: editable (Name, Description, Keywords, Adult) or not editable (Owner, Size, Type).
- For a Folder, these properties are editable (Name, Description, Keywords, Adult, Public/Private, Adult) or not editable (Owner, Size).

The information stored can be stored in some kind of Windows specific storage if there is a normal place to store it. **(Tom to specify)**. Whether or not such a place exists, the information should also be stored in an XML file index.xml in the Folder. This file follows the format of the V1 file, except that the top level should be <COLLECTION> rather than <COLLECTIONS> since there can only be one.

Note that the mime type MUST come from the Registry, V1 has a bug where myvideo.mpg will be mimetype=video/mpg instead of the correct video/mpeg.

The change in the Folder should be uploaded using POST Folder

If the Post Folder return code is 200 with a <THUMBNAILREQUEST>...</THUMBNAILREQUEST> response, then for each <FILE> mentioned, it should be passed to a Thumbnail generator (a component to be sourced from a third party). The JPEG returned should be uploaded to the server with POST Thumbnail.

Where other extended information is available, such as the Codec information Esc then Flycode should not interfere with them being visible.

Technology: Tom says: CPropertyPage - have code example.

It may be appropriate to code part of this functionality as a separate module - or as part of the FolderUpdater if it is necessary for the Properties dialog to meet certain Windows criteria **(Tom's input required).**

Folder Updater

This Module is responsible for keeping folders on the server in synchronization with Folders on the client. See Function: Folder Data Flow for how it integrates.

It is run whenever the client goes Online - as signalled by the Ping module, because Folders could have changed while the Client was off-line

The client reads http://valerie.flycode.com/intranet/spec/v2_architecture.htm#server_folder_hashes which gives a list of the Server's idea of the hash for each Folder.

The client walks the tree of Folders starting at Registry:FileClubs, and following any shortcuts, comparing the hash from "folder_hashes" with that in each folder's index file. This is the same directory walk as for the Indexer, and the Folder Updater can do this itself or request the hierarchy from the Indexer.

If there is no index file, then the Folder Updater should generate one from the information available from Windows about the file, and any Properties stored on it.

For any case where the hash doesn't match, or the Folder isn't listed then it can do a "POST Folder",

Note that it has to handle requests for thumbnails in the response (see Client:Properties)

Note that each "POST Folder" can be done by separate threads, running in parallel or it can be one thread that walks through and does them all.

Hash algorithm for collections (same as V1)

The collection hash is the XOR of the MD5 of the concatenation of the values from the fields sent with each file in a Collection. So algorithmically

- Start with a hash of all 0 bits
- For each item
    - extract the fields hash, type,adult,description,creator,file_size,filename

- Concatenate these fields and do an MD5 on them.
- XOR with the hash being generated.

Note that for an incremental operation consisting only of adding files, the new hash can be found by only taking the new elements and XORing into the oldhash.

Indexer

This keeps the Indexes on the server up to date. It is part of the Systray COM object.

There are three uses for this module:

- checking whether there is a local copy of a file,
- and incremental and full , both cases use the "POST index?oldhash=1a2b&hash=3c4d" command.
- caching the folder hierarchy - and making it available to other places such as the Add File dialog and Folder Updater.

The first time the application goes online after being launched then it will send oldhash="" to signify a full upload. Otherwise it will send the hash sent with the last POST.

The new hash is calculated by XORing the hash of each of the files with the oldhash. (If no oldhash is remembered then start with all A's i.e. "AAA....AAA"

If this is a full update, then the client walks the tree of Folders starting at Registry:FileClubs, and following any shortcuts. It builds an internal table of hash to location, that can be used for responding to GetFile's. (Note: Same directory walk as for the Folder Updater). The client can send each directory as a separate POST Index, BUT it should only send files not in its internal table, i.e. if the same file appears in several directories it will only be sent in one post.

If this is an incremental update of a single file (for example after a file is added to a FileClub), then the module checks if the file is already in the internal table, and if not sends a single POST index with the new file. If the client is offline then it should remember the changes and attempt to send when signalled by the Ping module that it is online.

If it is adding a directory to the index, then the module can just read that directory and send it in a POST index, it must check and only send files not already in the internal table.

This module needs an interface that allows the Uploader module and others to query for the location of a file by its hash.

The Indexer needs to log requests for files via a HTTP query to log_view

Synchronization

The Synchronization module should be run by the Launcher - it could also be run periodically - possibly under control of an Options field (Frequency to update Folders).

It walks the Folders under Downloads, checking their "Synchronize" property. If set (this is only set at the client), then it should direct the Folder Downloader to re-fetch the folder.

Folder Downloader

This downloads a Folder, given an id. It is a one-time action. It is driven by the Synchronization Module, and by the Download Dialog.

By default, it downloads Folders to Registry:Downloads/Funny Videos - fred

For a folder with id=123 it fetches http://server.flycode.com:2000/folder?id=123&hash=1a2b3c, where the hash is the last recorded hash from a previous call, and is got from the index.xml inside the Folder. A return code of 204 means that there is no change to the collection, so no downloading is required - this is used when synchronizing folders.

It should iterate through the folder fetching each file that it doesn't already have, through the PPH interface. For example, it can pass the PPH a URL of the form ptptp://server.flycode.com:2000/1a2b?name=coolvideo.avi&dir=Funny%20Videos%20-%20fred

If a different location is specified, then a symbolic link is required before downloading, for example if the user specifies D:/Funny then a link would be created at "Registry:Downloads/Funny Videos - fred" "D:/Funny" and the URL passed to the PPH is exactly the same.

Systray

The Systray component is part of the Systray COM Object which is launched by the Launcher, and is used to launch other pages.

This should have the following options:

- Transfers - this opens the Transfer Panel.
- Downloads - should open Registry:Downloads in a new window
- My Clubs - should open Registry:FileClubs in a new windows
- Search - Opens Search Panel in new window
- Home - Opens http://valerie.flycode.com/intranet/spec/v2_architecture.htm#server_redirecturls in a new window
- Exit - exits Flycode, it will be restarted by the Launcher if it is used.

UI issue: At this time I do NOT think we need a Connect / Disconnect option, this is because for receiving there is no concept of connecting, and we do not want to give the user an option to disconnect and NOT share, i.e. to get the benefits of Flycode without participating.

Note that several of these functions duplicate functions in the Flycode Toolbar, but I think this is worthwhile since they are available even when the user does not have an active Flycode window open

Listener

The Listener is a module that is part of the Systray COM module launched by the Launcher at startup.

It does a TCP/IP accept on a port specified in the registry.

When a command is received, the Listener parses the HTTP command and looks it up in a table of handlers. It should use some standard (e.g. something from Wininet) HTTP handler for this.

At this time, that table can be fixed, in the future there might be a registration process. Note that it must be possible to route GET and POST to different handlers.

Command	Handler
GET file	Uploader

When the handler is found, the Listener should pass on the HTTP connection to the handler, this should use as standard a Windows mechanism as possible - for example something like a ProtocolSink? **Tom to specify this.**

Uploader

An instance of the Uploader is created by the Listener for each incoming GET file command, receiving a HTTP structure **to be specified by Tom**

The URL will be of the form /file?hash=1a2b3c

The Uploader should query the Indexer to figure out where this file is located, and then should return the requested portion of the file via the HTTP connection.

If there is no matching file then it should return a 404 error code. This should not be a common occurrence.

The Uploader should notify the Transfer Manager via the same methods as the PPH uses.

There is no need to record state information since any retries are initiated from the requesting side.

There should be a limit on concurrent uploads (a Registry entry) and if this is exceeded then a 404 should be returned.

Transfer Manager

The Transfer manager works in cooperation with the Transfer Panel, they are separate because the Transfer Panel has a UI which is often not visible, while the Transfer Manager needs to run all the time.

See the Transfer Panel for details of how these two modules interact.

It is part of the Systray COM Object , when launched it will:

Check in the FlyTemp directory for incomplete downloads (1a2b3c.info files), since this module has to be running before the PPH is running it can safely assume that these are failed downloads - even if not marked as failed.

- If the registry entry Flycode/TransferSave = 0; delete all failed downloads - both 1a2b3c and 1a3b3c.info
- If Flycode/TransferSave = 1; It should restart all these downloads, calling PPH but without reading the data. The PPH will store the file in the Downloads directory, and communicate with the Transfer Panel.

Periodically (every 5 minutes) this module should run and check for failed downloads and attempt to restart them. How to recognize that a download has failed is to be researched, but probably it would work that if this module is already running (i.e. this is a periodic check, not a first-time check) then a 1a2b3c.info file will be correctly identified as "Failed".

There is functionality that is controlled by the Transfer Panel - for example Pausing or Retrying downloads, this could be either functionality of the Transfer Panel or of this Module.

Transfer Panel

This is opened by the Systray -> Transfers;

It receives events from the PPH instances as they download files. It also receives events from the Uploader . It works in cooperation with the Transfer Manager, which is part of the Systray COM Object and therefore running all the time. Some functionality could equally well be in this module or in the Transfer Manager, in general functionality that only happens when triggered by a UI interaction should be in this module.

When it is started, it contacts the Transfer Manager, the Transfer Manager then attaches it to all running PPH's, and the Uploader, for receiving events. When a PPH is created then the Transfer Manager is involved (for example it might be part of the class factory?) and can attach the Transfer Panel to it. When this module terminates it needs to inform all PPH's it is attached to so that it no longer receives events. (If it is easier, then it would also be possible for events to go to the Transfer Manager and then either be discarded or forwarded to the Transfer Panel.

The Transfer Panel should have the same functionality as now, specifically:

- When a download is initiated - by starting a PPH item, then an item should be added here, but this panel should not open unless clicked on in the Systray.
- When this module is started it should list all running PPH or Uploaders.
- Buttons in transfer panel
  - Delete should should delete an entry, and signal the PPH which should stop the download and remove the temp file.
  - Pause should mark as paused, and signal the PPH which should pause the download, and leave the file.
  - Clear Done should remove all completed downloads from the list.
  - Retry, should always get a file transfer going again, by whatever method is needed.
    - If an entry is paused, then mark as moving again and signal the PPH to continue the download.
    - If an entry is failed, then mark as moving again, and signal the PPH, or start a new PPH to continue the download.
    - If an entry is working, then signal the PPH to stop the download and try again.
    - Clear Uploads should only clear completed ones, and not effect in-process ones.
    - If the PC goes off-line, or is suspended, all downloads should be paused, when the PC is resumed, or goes back online (as signalled by Ping module) then all paused downloads should be restarted. **(Maybe this should happen in the Transfer Manager)**

Authentication

A number of commands are authenticated - see list - any module that uses these should be prepared to receive a redirection to a screen for entering login information. Commands will eventually be redirected back to the result.

Ping

The Ping module is responsible for informing the server that the client is still online

When started, and periodically (every 5 minutes) it should attempt to read (via a HTTP component) http://server.flycode.com:2000/ping (note this is lower case "ping", compared to "Ping" for V1).

If the Ping is unsuccessful then the client should be marked off-line, and the Ping module should keep trying, and the modules should be signalled.

If the Ping is successful then the client should be marked Online, and the modules signalled.

- The Folder Updater which will upload any changed collections - since the Folder Updater doesn't run continuously this means running it when you go online, and setting some variable, or having some interface accessable to the Folder Updater that it can check when it is run to see if it is online.
- The Indexer which will save pending changes when offline, and send any pending changes when it goes back online. So this should be signalled.
- Loggers of various types **(to be written)**
- Transfer Panel - or Transfer Manager - which pauses restarts when the client goes offline, and restarts any paused or failed downloads when it goes back online.

The first time it is called, and periodically (every 60 minutes) it should request http://valerie.flycode.com/intranet/spec/v2_architecture.htm#server_software_version. This should return a <SOFTWAREVERSION>. The client should check the serial number in the Registry, against the SOFTWAREVERSION parameters

- if the Registry < required then a dialog is presented, to the user offering a chance to download a new version or exit
- If the required <= Registry < available: then a dialog offers the opportunity to download or continue.

If the user chooses to Download then the client should run Active Install with a flag to indicate it is being run for an upgrade, and the client should exit.

Login

There is no "Login" module or functionality, it is handled by different modules.

- Telling the server where a client is, and what files it has is handled by the Indexer.
- Letting the server know that a client is still alive is handled by Ping.
- Checking the current software version is handled by Ping.
- Checking for <MESSAGES> from Send2Friend **is not specified yet.**
- Uploading information about Collections is handled by the Folder Updater

Registry

There is clearly a design issue here, the question is whether we have a one-to-one correspondance between Flycode users and Windows users. If we think there is, then we put shared Registry entries in HKEY_LOCAL_MACHINE/Flycode and per-user entries in HKEY_CURRENT_USER, but if we think that Flycode users may use a single login, and then setup multiple Flycode users underneath that, then we need to keep something like the current structure.

The assumption is, that if they are sophisticated enough to setup seperate Flycode users then they can do it by having separate Windows logins, and so we can do this split.

The registry is organized as:

Path	**Key**	**Default Value** **(... means C:\Flycode)**	**Read**	**Write** (unless spec. written at installation)	**Meaning**
KEY_CLASSES_ROOT					Settings related to intercepting protocols, types etc.
					Tom to document.
HKEY_CURRENT_USER\Software\Flycode					All settings personal to this user
	Downloads	....\mitra\Downloads	PPH, Explorer Bar, Folder Downloader, Systray		Root for storing downloads and downloaded clubs
	FileClubs	....\mitra\FileClubs	Systray; Folder Updater; Indexer;		Root for storing user's own clubs
	FlyTemp	...\mitra\FlyTemp	PPH, Transfer Manager		Root for temporary files, note this must be on the same drive as Downloads
	Users	...\mitra\Users	Explorer Bar, Right-Click/Add to Hotlist.		For the hotlist
	Searches	...\mitra\Searches	Explorer Bar, Search		For the result of searches
	TransferSave	1	Transfer Manager	Options	Save transfers and restart when possible
	SearchAdult	2	Search		Value from last search
	SearchVideo	1			Value from last search
	SearchImage	1			Value from last search
	SearchFolder	1			Value from last search
	SearchOnline	1			Value from last search
	UploadLimit	5	Uploader		Limit on number of simultaneous uploads
	AddFileClubMove	0	Right-Click Add FileClub		1 means move, 0 means copy 2 means create shortcut.
	AddFileClubClub	""			Club to offer first, defaults to Registry:Fileclubs
					Settings common to all

HKEY_LOCAL_MACHINE\Software\Flycode					users of Flycode on this machine
	AppPath	C:\Program Files\Flycode	?		Where the application is installed
	Version	?	**About**		Current build number
	ListenerPort	3301	Listener		Where we listen for incoming requests

Questions:

- Do we need a HKEY_LOCAL_MACHINE\Software\Flycode\Default that is used to load the initial settings for a new user?

Directory Structure

The client directory structure is intended to be simple and easy to work with. There is an important distinction between the logical structure of Flycode as presented in the Explorer Bar and the physical structure on disk, below is the default structure, but various parts can be moved either by changing the registry keys or by using shortcuts, normally this would be done to allow large Folders or Files to be one other parts of the disk than C:, or to save having two copies of something. *Italicised entries are examples, replaced by the user id, or club name etc.*

C:/Flycode	root of flycode hierarchy,
jsmith	directory for a particular user
Downloads	where downloads for that user are stored. [HKEY_CURRENT_USER\Software\Flycode\Downloads]
Fun Videos - Mitra	a club downloaded from *Mitra*
Big Videos - Jane	a shortcut to a downloaded club from Jane somewhere else on disk.
FileClubs	where the user's own fileclubs are stored. [HKEY_CURRENT_USER\Software\Flycode\FileClubs]
India	A specific club
elephant.mpg	A file in the club
bigvideo.avi	A shortcut to a video stored elsewhere on the disk.
Huge Club	A shortcut to a FileClub stored elsewhere on disk
FlyTemp	the temporary area used for downloads, this MUST be on the same drive as Download - [HKEY_CURRENT_USER\Software\Flycode\FlyTemp]
Users	area for storing a list of specific users [HKEY_CURRENT_USER\Software\Flycode\Users]
Mitra	internet shortcut to http://server.flycode.com:2000/user_folders?userid=*mitra*
Searches	Area for storing results of searches [HKEY_CURRENT_USER\Software\Flycode\Searches]
funny.xml	Results of a search for "funny"
C:/Program Files/Flycode	where the application is stored [HKEY_LOCAL_MACHINE\Software\Flycode\AppPath]

Right-Click

If possible a Right-Click menu will be present on all links in a Folder. This will contain.

Folder	File inside Flycode	File in Windows/I.E.	User inside Flycode	Text	Does
	*			Find Folders containing this image	Navigate to http://server.flycode.com/folders_containing?hash=1a2b
		*		Add to Flycode	Open a dialog with a nested list of Flycode folders shared on this machine, when a Folder is selected acts as if file was dragged into that Folder. This could also be a cascading list attached to the right-click menu. If its a dialog box it should give a choice of Copy/Move and also remember the last Fileclub chosen.
*				Download Folder	Download Dialog
*	*			Properties	Opens Properties dialog with Flycode extensions. (Read only if Remote Folder or File in Remote Folder)
			*	Add to Hotlist	Adds the user to the Hotlist folder as an IE shortcut - see Directory Structure.
			*	View User's Files	Navigate to http://server.flycode.com/user_folders?userid=jsmith (see Client:Folders_XSLT)

Technology: Tom says can add arbitrary properties to the right-click context menu (like Winzip) - have code example

Options

(only options for parts of code implemented are required by 20th May)

Many parts of Flycode could potentially be configurable.

The goal of options is to allow the more advanced users to have some control over the application, while protecting beginning and intermediate users. This will eventually be done by having a main panel which has a course level of control - e.g. setting defaults based on bandwidth, and then other panels with more fine control. For now its a single panel.

This should be a "Options" button on the Toolbar, or menu. **(Tom to figure out exactly where it goes)**

The button should open either an HTML form, or a dialogue.

- This box should be multi-tabbed. The left-most, default tab labelled "General" contains items to set defaults based on line speed. it would have items on it which said:
  - ☑ Start Flycode with Windows
  - Size of cache - no more than 1Gb ▾ *<initially set to 10% of disk space vailable>* (Above this size and the Transfer Manager should start emptying the cache)

- ☑ Save Transfers - should set FlyCode/TransferSaves = 1 or 0

A review should be made of the client code to determine if there is anyting else that it makes sense to configure.

This should set Registry entries as it does in V1.

Open issues

Not in V2.0 - Questions for Tom - Compatibility issues - Email Notes

Not in V2.0

This list is items mentioned in the Product Description, but that I don't think should be in V2.0, better V2.1, these are suggested because the feature is relatively unimportant AND is complex to implement.

- Everything under Publisher/V2.0/Subscriptions except basic pay a fee to get access to content for that
- Publisher/V2.0/Publishing/Background fingerprinting
- User/V2.0/Searching/Number of copies available
- User/V2.0/Folder/Graphic display of License information e.g. bar showing days till expiration
- User/V2.0/Transfers/Advertising in Transfer window
- User/V2.0/Transfers/Start playing in WMP immediately on downloading.
- User/V2.0/Transfers/Tallies - I'm not sure what is meant by this.
- User/V2.0/Community - everything
- User/V2.0/Viewing:Printing and Wallpaper
- Tallies for uploads/downloads: For each local file on your disk, metadata showing 1.) The number of times the file has been uploaded from you. 2.) The number of times the file has been downloaded in the entire Flycode network. Indicates popularity of files.
- Storyboard/Start/Customized start page
- Storyboards/User Profile
- Storyboards/Threaded Discussions
- Storyboards/Search History
- Storyboards/Recommend
- Storyboards/View/Blacklist
- Storyboards/User Hotlist - as a separate panel, just show in the Explorer bar.

Questions for Tom

- I need to know if when a .mbox file is launched, and goes to the MLM, then will it then go automatically to WMP or will we see a MLM window
- Can we grab Windows failures to launch, so that we can use them to look for a suitable player

Compatibility Issues

- We will assume IE 5.0 or higher
- We should work with IE6.0 betas
- We need to run under: Win98 2nd edition, Win2K, WinME, and as it becomes available Win XP.

Notes from Email

These are some collected notes from email that might be useful, they do NOT constitute part of the specification, if they conflict with anything else in this document then discount what is said here, or check with Mitra.

- Tom: 28 Mar
    - Can add custom columns in File Explorer - have code example
    - Can customize icons file-by-file, for local, remote, DRM'ed, etc. Have code example.
    - Hook handler for moving/copying/deleting/renaming folders - have code example.

Useful references

- XML, XSL and XSLT
    - Displaying XML in Internet Explorer shows basics and how to refer to a XSL from a XML
    - Unofficial MSXML XSLT FAQ useful reference, especially if things don't work as expected.
    - What is XSLT - extracts from a book, more comprehensive, but longer
    - Transforming XML - a Folder of documents on doing XSLT
- Pluggable Protocol Handlers
    - Asynchronous Pluggable Protocols Overview
    - Or search for articles on MSDN
- Extending Windows
    - Shell Extensions from MSDN
- Other programs to look at for examples:
    - Internet Neighborhood. - Custom Shell supporting drag and drop for remote and local files.
- Thumbnails
    - Alta Vista Image Search
    - Microsoft's Thumbnail control

APPENDIX D

Flycode
Database Specification

Flycode
Database Specification

(Company Confidential)

Last updated: 28 May, 2001 by Mitra

Last checked for correctness: 1st May 2001 by Mitra

This documents the V1 Server operating, it needs the database_delta spec integrating when that goes live. And also changes in the v2_architecture spec. There are also proposed changes in v2_after and mediatypes and private_label.

Purpose

The purpose of this document is to provide a design specification for the database structure as implemented in the Flycode server. The database specification is intended to mostly independent of the Client->Server protocol and totally independent of the user-interface, however certain compromises may be made for efficiency.

Technical Control

This design and the control of the software specifications are being managed by:
Mitra, Flycode Chief Technology Officer. <mitra@earth.path.net>

All questions relating to this specification document should go to him.

Technical Control

This design and the control of the software specifications are being managed by:
Mitra, Flycode Chief Technology Officer. <mitra@earth.path.net>

All questions relating to this specification document should go to him.

Technical Control

This design and the control of the software specifications are being managed by:
Mitra, Flycode Chief Technology Officer. <mitra@earth.path.net>

All questions relating to this specification document should go to him.

Specification Index

Entity Diagram

Tables described elsewhere in scheduled development.

- In Process
  - betarequests
  - viewlog
  - messagelog
  - ratings
  - layout
  - layoutparams
  - sessions
  - copyrights
  - group
  - members
  - requests
- Private Label
  - network
- Preview Integration
  - offerfiles
  - offers
  - owners
  - conditions
  - receiptsplit
  - optinemail
    optinusage
    optinmessages

Technical Control

This design and the control of the software specifications are being managed by:
Mitra, Flycode Chief Technology Officer. <mitra@earth.path.net>

All questions relating to this specification document should go to him.

Overview

The tables of the Flycode database can be considered to occur in three groups.

1. Application Tables: providing the functionality of Flycode
2. Log Tables: These accumulate information about what has happened, they can be truncated without damage to the application, although of course this should be done AFTER the information is analyzed.
3. Statistical tables: these will be used to monitor the use and growth of Flycode. This set of tables might well change a fair bit as we gain experience with the product. Generally these tables will need a various cron type tasks defined so as to populate them. This functionality is not critical to the Alpha development of the product, but needs to be there for launch.

It is critical that anything using these tables should assume that they might change in the future, in particular all code that relates to maximum size of strings should be configured from header files so that the string lengths can afterwards be changed in only a few places (e.g. Database creation routine **where?**, Server: DBClasses.DBConstant and Client C++ config file **where?**)

Contents:

Application Tables

Abbreviations

Abbreviation	Means	Comment
uint	Unsigned Integer	10 bytes

datetime	BigInt20	Holds a Unix timestamp
USERID	CHAR40	Long enough for a Flycode userid
EMAIL	CHAR40	Long enough for any reasonable legal email address
HASH	CHAR22	Holds a Base64 encoded MD5 value
IPPORT	CHAR8	IP number and port expressed in hex
FILENAME	CHAR255	File name

The following database tables will be used in the design.

configuration

This is a table used to contain various configuration values used by the software. It has four columns:

Column	Datatype		Description/Comments
rowid	uint		autoincrement
valuename	CHAR25	UNIQUE	the name of the value in this row
value	CHAR80		the value associated with the valuename
description	CHAR64		the description of the valuename

Specific Rows that are needed in this table should be documented here. Please email mitra if you add any rows to this table.

Valuename	Example Value	Description
DBVERSION	1	Version number of database, increment each time a field is added/deleted to ANY table
SWMINVER	220	Minimum version number of server that this database is compatible with. Note that the server version number and minimum version of database are compiled into the server, as to put them here would defeat the point of making sure they server code and database are commutable.
AVLCLVER	430	Currently available version of client
MINCLVER	410	Minimum version of the client required to work with the Server.
SETUPURL	http://..../flycodesetup.exe	URL for new version of active install
SETUPVER	7	Active Install Version

size 181 bytes per entry

users

This table contains the list of Flycode registered users. An entry is made in this table when a new user registers to use Flycode. On subsequent logins and logoffs the online column and location columns are updated. A process needs to run from time to time to update the table for disconnected users who did not logoff.

See Modification Scheduled in InProcess: LastSeen

Column	Datatype	Description/Comments
rowid	uint	autoincrement
userid	USERID	user's login id
password	HASH	Hashed password: The value MD5(userid ':' realm ':' password) ; note that this is the same information as generated by /usr/sbin/htdigest as supplied with Apache.
connection_speed	CHAR10	from initial registration
online	uint	1 if on-line, 0 otherwise
time_created	datetime	Unix timestamp when they registered
location	IPPORT	IP Number and port expressed in hex
authenticate	CHAR150	Cached Authentication response
email	EMAIL	Email address of user
port	uint	port to contact the client on
server	uint	index into servers table of the server handling this user
thread	uint	thread number (or other identifier) on that server.
notice	uint	frequency in days requested for notification
time_left	datetime	Unix timestamp of when disconnected, set to 0 during session or after cleanup.

Total Size 370 bytes per user

emailaddresses

This table contains the email addresses that a user has. When a user registers, one email address is added to this table. Subsequently (as a 'preferences' capability on the client) the user can add additional e-mail addresses. This requires a suitable message package and a server processing of same. When Flycode is installed, a validation code is generated and passed to the client. To receive messages, the member must validate by entering this code. When that is done the validationflag is set to 1

Column	Datatype	Description/Comments
rowid	uint	autoincrement
userid	USERID	user's login id
validationflag	CHAR1	0 when Flycode installed, 1 when member has validated their email address
email	EMAIL	Email address of user
validationcode	CHAR10	Used during validation process

Total size 101 bytes per email address

userfiles

This table contains the media files that are currently available on the Flycode network, identified by users that have them. When the user logs in the index is uploaded into this table; on log off your entries are purged. If a user attempts to access a file that is no longer connected, an error is generated

See In Process / Record User files by IP for changes to this.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
hash	HASH	hash value of the media file
userid	USERID	user name of user having this file (note this changes to sessionid with changes to user IP rather than user.)
time_entered_online	datetime	connect timestamp
connection_speed	CHAR5	from users file

Total Size 97 bytes per file online

files

This is the table of Flycode media files. Whenever a new file hash is loaded in an index, a row is added to this table.

Modification Scheduled in InProcess: LastSeen

Column	Datatype	Description/Comments
rowid	uint	autoincrement
hash	HASH	hash value of the media file
mimetype	CHAR20	image/jpeg, image/gif, etc.
adult	CHAR1	flag 2 = adult 1 = unmarked, 0 = family friendly
copyright	CHAR1	flag 0 = OK 1 = suspended 2 = copyright 3 = illegal
filesize	uint	in bytes
thumb	HASH	hash value of the media file's thumbnail, if any
notransfers	uint	the number of transfers of the media file
lasttransfer	datetime	timestamp of most recent transfer
private	CHAR1	1 = by invitation, only , 0 = public
creator	USERID	The creator of this file where known

Total size = 167 bytes

comments

This table contains comments contributed by viewers on media files.

See proposed change for Private Labels to add "network" field.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
hash	HASH	hash value of the media file
timestamp	datetime	timestamp when loaded
comments	CHAR255	the comment entered by the member

userid	USERID	username from login of the commentator
collectionid	uint	optional -- only if in collection

Total size 357

collections

This table is a list of media collections (file clubs), identified by owner. This is one of the tables that is searched when a search query is processed. This table is searched first and if the max number of hits is not achieved, then the elements table is searched

See Modification Scheduled in InProcess: Content Serving

See proposed change for Private Labels to add "network" field.

Column	**Datatype**	**Description/Comments**
rowid	uint	autoincrement
userid	USERID	username from login
name	CHAR60	file club name
description	CHAR255	description of file club
private	CHAR1	flag 1 = by invitation, only , 0 = public
timestamp	datetime	timestamp when loaded
hash	HASH	A hash of the values sent with the last Upload_Collection
size	uint	Count of elements for this collection
imagecount	uint	Number of images in the collection
videocount	uint	Number of videos in the collection

Total Size 438 per collection

elements

This table contains the descriptions for a media file as added by various file club owners of that image. This is one of the tables that is searched when a search query is processed.

Column	**Datatype**	**Description/Comments**
rowid	uint	autoincrement
collectionid	uint	rowid in collection table
filename	FILENAME	name of file as it appears in this collection
time_added	datetime	time at which the fileclub element was added
hash	HASH	hash value of the media file
description	CHAR255	file club owner's description of media file
collections_private	CHAR1	1 if the collection is private, and therefore the elements should not be searched.

messages

this table stores messages that are being sent to a friend. The number column is decremented when a message is sent. When it zero the message is deleted. A message is added to this table if it can not be delivered at the time the server processes it. Messages to e-mail addresses will be sent automatically by the server so this only applies to messages to Flycode users that are not logged on at the time the message is sent to the server. A purge routine should remove messages after some period of time.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
username	USERID	user's login id of message sender
sendto	CHAR255	list of recipients (Flycode and email) in text form
subject	CHAR80	subject string sent from client
messagestring	CHAR255	message string sent from client
time_added	datetime	time at which the message was added to the table
hash	HASH	hash value of the media file being referenced in the message
number	int	the number of messages remaining to be sent
name	FILENAME	the name of the file
collectionid	uint	optional - if there is an identifiable collection associated with this message

Total size 733 bytes per message

messageemail

Records emails addressed to non-apple soup users.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
messagenumber	uint	the rowid of the message in the messages table
sendto	EMAIL	email address of this recipient
time_received	datetime	time at which message (not the email message, the web page) read by the recipient- initially set to 0
seed	CHAR10	initialized to a random value when the record created

Total size 90 bytes

messagelist

the list of addressees for a message. When a user logs on a search is done of this table and those messages that appear are delivered. The corresponding rows on this table are deleted and the counts in the messages table are decremented and a record added to messageslog. (NB this requires a message structure to exist on the client to store such messages until they are viewed

Column	Datatype	Description/Comments
rowid	uint	autoincrement
messagenumber	uint	the rowid of the message in the messages table
userid	USERID	userid of the Flycode user that the message is for
reminded	datetime	When a reminder was sent via email (typically 48 hours after message send)

Total size 80 bytes

feedback

This table contains answers from responses to forms - a set of answers from a single form or set of forms is grouped by a response_id. The table is filled by the JSP code behind pages like feedback.htm, question_id comes from the NAME field of a form, and answer is the result (URL encoded). Note that the forms may include hidden fields set by parameters in the URL, such as for example the user's Flycode id.

See proposed change for Private Labels to add "network" field.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
time_added	datetime	time at which the transfer took place
response_id	HASH	same for all queries during same session with web server
question_id	CHAR40	id given in <QUESTION ID attribute
answer	CHAR255	answer as entered by the user.

Total size 347 bytes

language

contains multilingual strings for use by the server, the server will look for example for the English version of string "welcome". Currently there are no strings sent back by the Flycode Server to the client, so this table is only used by the JSP pages.

See proposed change for Private Labels to add "network" field.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
stringid	CHAR20	a message id
language	CHAR20	language of string as specified in rfc1766 which refers to ISO639 and ISO3166 e.g. "en" or "en_US"
text	CHAR255	text of string in UTF8

string+language is a composite key.

Total size 305 bytes

viralemail

This table lists each email address that is used in the messaging system. The first time the email address appears a new row with that address is added with the inittime & recenttime time stamps being set, and regtime being null and the dontsendflag being set to 0 and count to 1. Each time a subsequent mail is sent to this same address, the recenttime datetime stamp is set with that time and count incremented. If the address is registered as a Flycode user, the regtime timestamp is set. If mail to this address bounces back for invalid address, the dontsendflag is set to 1 and a message sent to the sender. If the recipient sends a don't send message the dontsendflag is set to 1. (note: may want to refine this so that more than one not delivered is required before it is removed.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
emailaddress	EMAIL	the email address to which the message is sent
dontsendflag	CHAR1	0 if okay to send mail to this address (default), 1 if recipient declines receiving or invalid
inittime	datetime	time at which first email was sent to this address
recenttime	datetime	time at which the most recent email was sent to this address
regtime	datetime	time at which the recipient registered a copy of Flycode
count	uint	incremented each time a message is sent

Total size 121 bytes

livesubscriptions

contains a list of which currently online users are subscribed to which collections. These entries are removed on logoff.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
userid	USERID	userid of subscriber
collectionid	uint	Rowid of Collection
hash	HASH	hash of last version seen by user.

Total size 152 bytes per subscription

servers

contains any information that needs to be kept across servers. Note that the rowid is used as an index in other tables. This table will be expanded to hold any information that needs to be accessible across different servers, for example to an administration program.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
IP	IPPORT	ip address and port in hex

dbcheck

contains information that is used in performing consistency checks on the database

Column	Datatype	Description/Comments
rowid	uint	autoincrement
foreignkeytable	CHAR25	a table name in this database
foreignkeycolumn	CHAR25	a colum name from that table
referencedtable	CHAR25	a table name in this table that the entry in the foreignkeycolumn points to
referencedkeycolumn	CHAR25	the column which entries in the Foreignkeycolumn point to.

exedataplat

Column	Datatype		Description/Comments
rowid	UINT	NOT NULL	AUTO INCREMENT, PK
majorversion	INT		Version of Flycode client application Flycode/<major_version>.<minor_version>.<build> <platform> e.g Flycode/1.0.450 PC
minorversion	INT		
build	INT		
platform	CHAR10		
exedataid	INT		Checksum of the block

exedata

Column	Datatype		Description/Comments
rowid	UINT	NOT NULL	AUTO INCREMENT, PK
cs0	INT		
cs...	INT		
cs99	INT		

100 Rows will be inserted (per platform) in this table while making client build live.

See the server spec for how this data is used. This data is precalculated during the client build.

In the future, we can further optimize this process by making separate tables per platform

Log Tables

transferlog

This table is the list of successful transfers. Note that userid is NOT recorded.
See proposed change for Private Labels to add "network" field.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
time_added	datetime	time at which the transfer took place
hash	HASH	hash value of the media file
demographics		fields will be added here to record demographics to be specified by Cate, these will be obtained from the users table, but userid will not be recorded

Total size 52 bytes

searchlog

This table logs searches. Note that userid is NOT recorded.
See proposed change for Private Labels to add "network" field.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
time_added	datetime	time at which the search took place
term	CHAR80	
number_results	uint	number of results returned

Total size 120 bytes

transferfailurelog

This table logs transfer failures. Note that userid is NOT recorded.
See proposed change for Private Labels to add "network" field.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
timestamp	datetime	time at which the search took place
hash	HASH	hash of file that was being transferred
ip	IPPORT	IP number and port (from user's table) that was being contacted
reason	CHAR40	reason for failure - this may have internal structure to be defined.

Total size 100 bytes

Statistical Tables

The following tables are used to monitor various aspects of the usage of the database. They are generally populated with snapshot data which are recovered by processes that are spawned by a scheduled (e.g. cron) process.

See inprocess.htm for changes to this

logonsession

a record of the logon sessions

Column	Datatype	Description/Comments
rowid	uint	
userid	USERID	userid
logon timestamp	datetime	the time at which logon occurred
logoff timestamp	datetime	time at which the logoff occurred

Total size 90 bytes

numberofdownloads

gives the number of downloads to that time. This is a process that goes to the download site(s) and registers the count at those sites at the time

Column	Datatype	Description/Comments

rowid	uint	autoincrement
value	uint	the count
site	CHAR10	the download site
timestamp	datetime	time at which the count was taken

total size 50 bytes

counters

This is a generic table for counting something for statistical purposes.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
id	CHAR20	an id
count	uint	the count
timestamp	datetime	time at which the count was taken
description	CHAR80	Description of the counter - e.g. who is using it, or what it is counting

Total size 140 bytes

This table is used to record an incrementing, or periodic count. A process wishing to increment the count for id=foo would increment the count for the record with id=foo and timestamp=0 (or null or whatever). Alternatively a process just wishing to record a count can create a new record with a specified id and current timestamp.

Periodically - and this might vary based on id - the server will set the counts, and create new records with timestamp=0.

Consistency Checks

The following consistency checks can be applied periodically to check the integrity of the database.

- All of {livesubscriptions:userid, livesubscriptions:owner, etc.} exists in users:userid
- Foreach collection in collections { collection:size = count(elements: elements:collectionid = collection:rowid) }
- more are to be defined here.

APPENDIX E

Provisional U.S. Patent Application No. 60/212,177 Entitled: *File Distribution and Storage Apparatus and Method Over a Global Network*

Provisional U.S. Patent Application

Inventors: Scott, Adrian C.H. , San Francisco, CA
Assignee: AppleSoup, Inc.
Filing Date: June 16, 2000
Attorney: Michael Louie

Abstract

A distributed storage and transfer system for audio, image, video, and other data files. Files are stored in distributed clients and are accessed through directory information that resides on central servers.

Background & Description of Invention

Existing architectures for file distribution store the files on central servers (Figure 1), which are accessed by individual client programs.

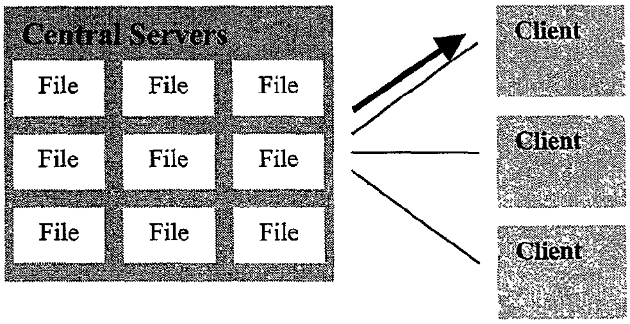


Figure 1:
Centralized File Storage and Transfer Architecture, in which files are stored in central servers and accessed by distributed clients

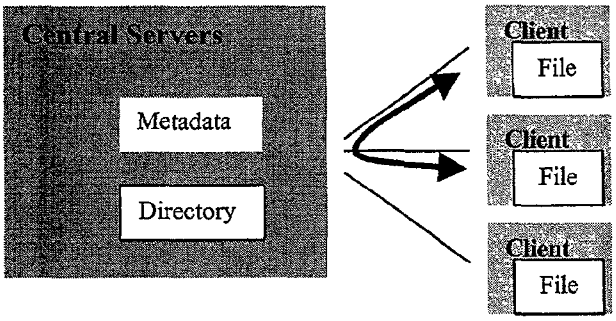


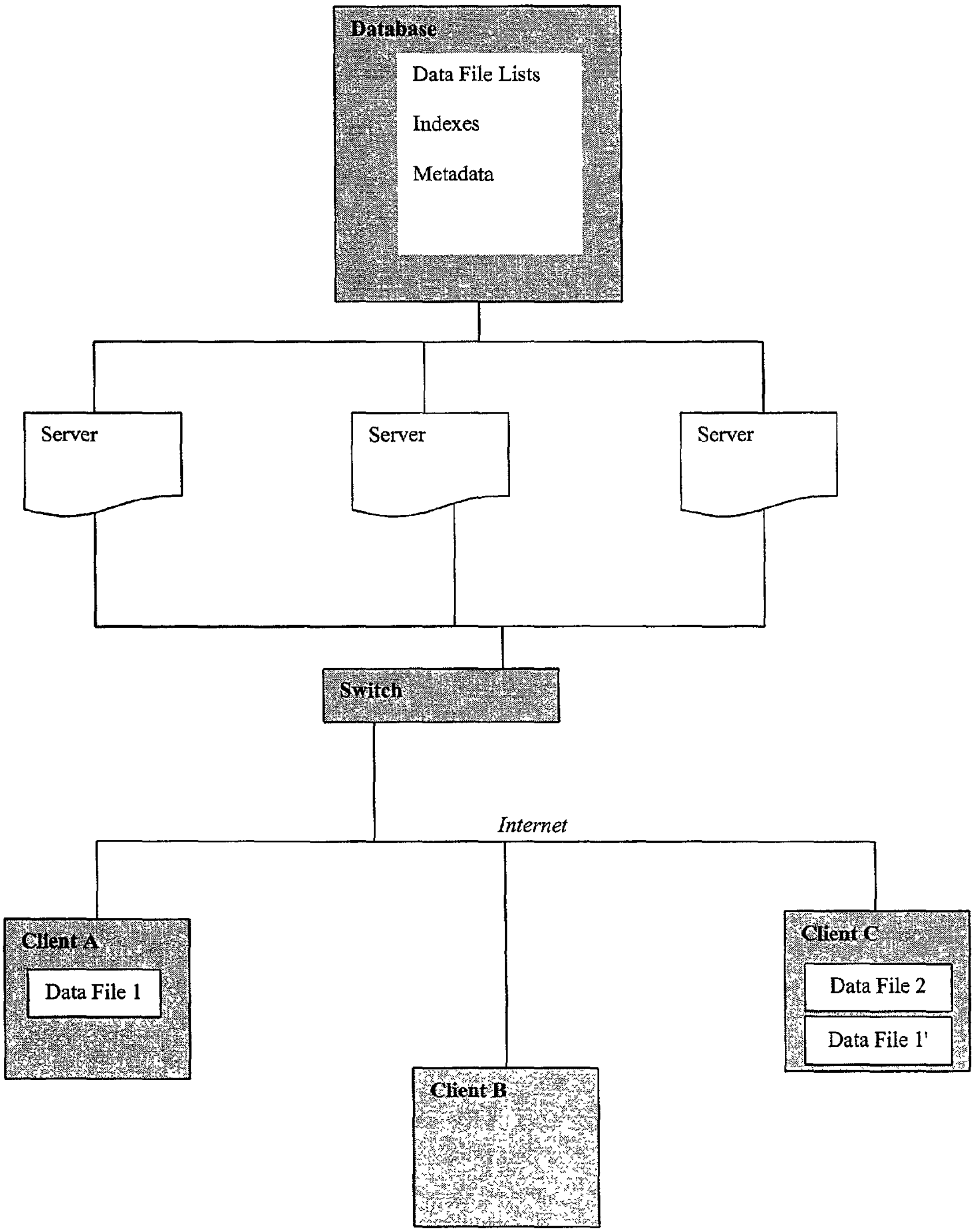
Figure 2:
Distributed File Storage and Transfer Architecture, in which files are <u>stored on distributed clients</u> and transferred between them via the central servers

The present invention stores individual data files on distributed client systems, with directory and metadata residing on the central servers (Figure 2). Files are transferred directly from client to client through connections made by the central server. This system has numerous advantages over the prior art, including greater scalability and economical data storage.

Figure 3, below, displays a diagram of the preferred embodiment of the inventions claimed and serves as a reference used in descriptions of the workings of the inventions.

Distributed file storage:
Data files are stored on the individual machines comprising the network of distributed **clients**, rather than on the servers of the system itself.
The **servers** to which the clients connect are in turn connected to a single **database** containing Metadata information about the data files that are known to the system, but these servers do not contain the data files themselves, which remain resident on the distributed client machines.

Figure 3:
Architecture of Preferred (AppleSoup) embodiment, in which files are stored on distributed client systems and transferred between them via central servers.



File Identification & Search:
Identical files are recognized as such through the creation of file 'fingerprint' identifiers, accomplished by using hashing functions such as MD5 to create identifier strings which uniquely correspond to the data contained within the data files.

Clients which connect to the system make the files on their machines visible to and available to other clients who may also be connected to the system. The master **list of data files** which are known to the system is located on the servers and can be searched by any client which is connected to the system.

For example, in Figure 3, **Client B** connected to the system sees that **Data files 1 and 2** are available for transfer. **Data file 1** is available from two different sources, **Client A** and **Client C**, where in the latter case it has been renamed to **Data file 1'**. Using the system's file identification algorithm, is still recognized by the system as being the identical to **Data file 1**, and is therefore treated as such.

File Sharing:
Client B may request a transfer of any data file known to the system to its host machine. For example, the user at **Client B** may request the transfer of **Data file 1** to its machine. That request is processed by the server, which, on the basis of certain parameters (such as transmission speed, other file transfers in effect, available bandwidth, geographic location, and client identifier) *automatically selects the source client from which the transfer is to be initiated* --in this **case Client C** --and permits the two clients to transfer the file between them. The **Data file lists** stored on the central server are updated to show that **Data file 1** is now located on **Clients A, B, and C**.

If the transfer of **Data file 1** between **Client C** and **Client B** is interrupted for any reason, such as **Client C** being disconnected from the network, the system automatically resumes the file transfer using the copy of **Data file 1** residing on **Client A**. The transfer is resumed at the point of interruption, so that the portions of the file which have already been transferred to **Client B** do not need to be re-transmitted. Furthermore, if the interruption is caused by the disconnection of **Client B** itself, the data file transfer is automatically resumed from a valid source once **Client B** re-connects.

Simultaneous downloads:

Identification of identical files permits clients to download a data file from multiple source locations simultaneously, with different portions of the data file being received from different locations and then re-assembled at the destination. Thus in the transfer above, the first half of **Data file 1** might have been transmitted from **Client A**, and the second half from **Client C**, with the pieces properly re-assembled at destination **Client B**

Distributed file hierarchies:

Data file indexes and hierarchies are created by each client on its host machine as an AppleSoup fileclub and then unified into a single structure on the central database. Thus if **Data file 1** on **Client A** is classified into a fileclub called 'Pets', and **Data file 2** on **Client C** is classified into the same fileclub, then any clients connected to the system would see both data files appearing under the Pets fileclub on the central **data file listings**.

Client Authentication:

When a client attempts to connect to the server the attempt is allowed only if the client is identified an AppleSoup client. This authentication is achieved by creating a '**session challenge key**' on the server which is passed to the client together with an authentication request. The **session challenge key** is created by using the MD5 hashing function on a string of data that includes information such as the time and a location identifier. This session valid key must be correctly interpreted to return a **response** which is created by taking an MD5 hash function value on a string of data which includes user identification, other data passed by the session valid key and a portion of the client software code which portion is identified in the **session challenge key** location identifier. Since the location changes randomly, this effectively prevents the connection of any client software which is not identical with AppleSoup's client software.

Database Integration:
The preferred embodiment system uses one single database for its metadata and directory information. This database is accessed by all servers. Alternate embodiments may used multiple databases which may be accessed independently of each other by the servers.

The preferred embodiment is more fully described in the following attachments:

- ➢ Client specification attachment
- ➢ Server specification attachment
- ➢ AppleSoup Database specification attachment

Claims

A distributed data file storage and transfer system comprising:

- ➢ **Distributed file storage**: Data files are stored on distributed clients. *See Server Specification, Get_File.*

- ➢ **Central repository for metadata**: Only file metadata identification and descriptor information (e.g. name, location, size, comments, captions, and other data fields) is stored on the central servers. *See AppleSoup Database Specification Attachment, files table specification.*

- **File identification**: Files are uniquely identified by file signature function, where the file signature is the unique index identifier of the file. This is accomplished by performing MD5 on the entire file, creating an identifying string. This procedure allows files which differ only in name to be correctly identified as the same, and simultaneous piecemeal transfer of a file from multiple sources. *See Server Specifications Attachment: Media file Hash function.*

- **Source file auto-selection**: Source file selection is performed automatically by the system, (from the available source files). *See Server Specification, Get_File.*

- **Auto-reconnect**: Interrupted transfers are resumed automatically upon reconnect, resuming transfer at the point of interruption. *See Server Specifications Attachment: Connection Loss.*

- **HTTP protocol**: Distributed client system architected and implemented using HTTP protocol. *See Server Specifications Attachment: Communications Protocol.*

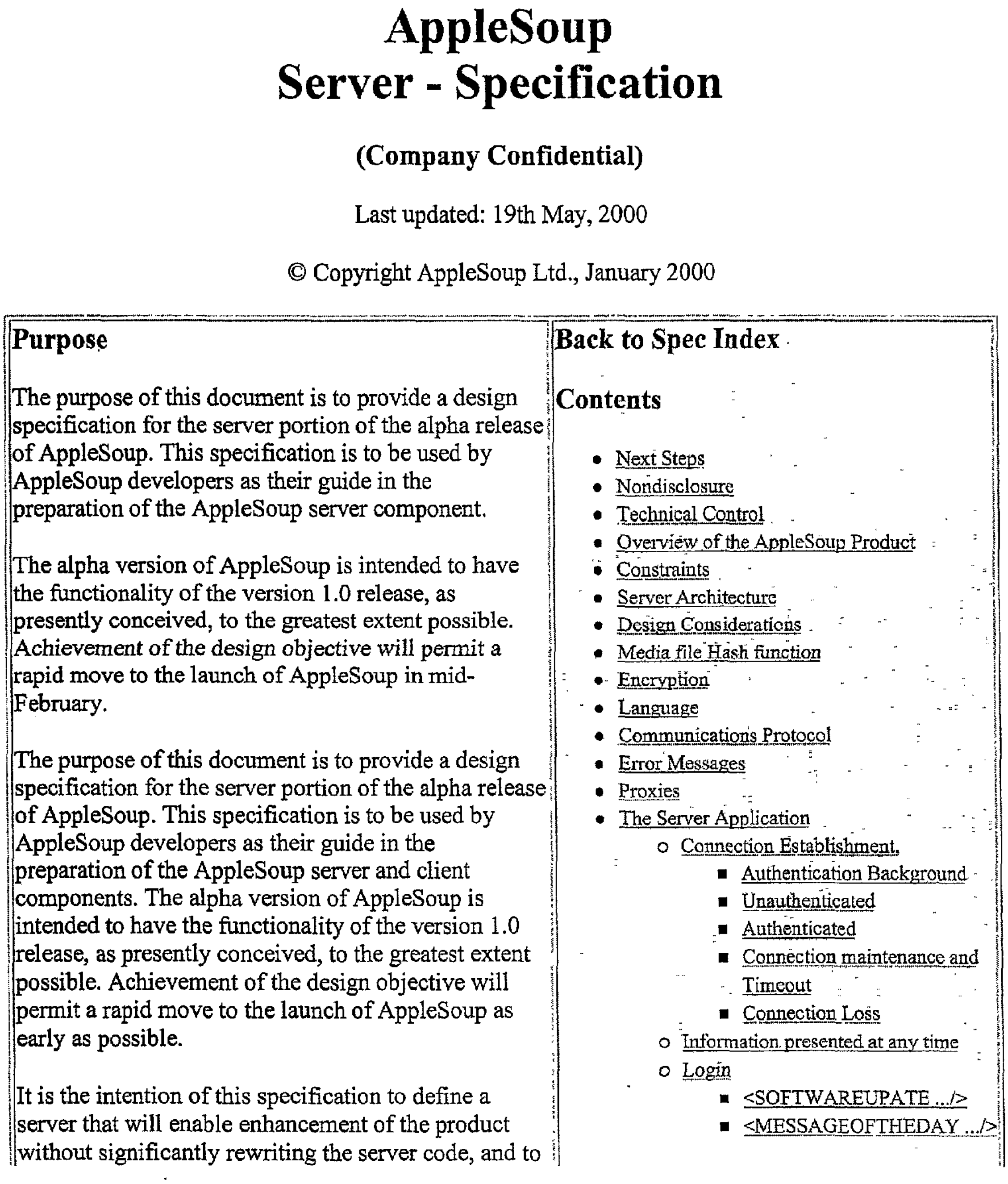
- **Single database**: Distributed client system architected to a single database, which stores all metadata and directory information and which can be accessed by multiple servers. *See AppleSoup Database Specification in its entirety.*

- **Distributed database**: Distributed client system architected using distributed databases, where servers may access different and distinct databases.

- **Distributed indexes**: Distributed file indexes and hierarchies. File hierarchy is created by the distributed client systems and unified on the central servers. *See Client Specification, File and Data Storage Structures.*

- **Client authentication:**. Access to the functionality of the AppleSoup system is only available through AppleSoup client software. The client authentication key includes an electronic fingerprint of a randomly selected portion of the client software as part of the identification key used in a given connection session. *See Server Specifications Attachment: Connection Establishment and Authentication and Loss.*

- **Simultaneous Downloads:** Data file transfers can be executed with different segments of the file coming from different client source locations. *See Server Specification, Get_File.*



AppleSoup
Server - Specification

(Company Confidential)

Last updated: 19th May, 2000

Purpose

The purpose of this document is to provide a design specification for the server portion of the alpha release of AppleSoup. This specification is to be used by AppleSoup developers as their guide in the preparation of the AppleSoup server component.

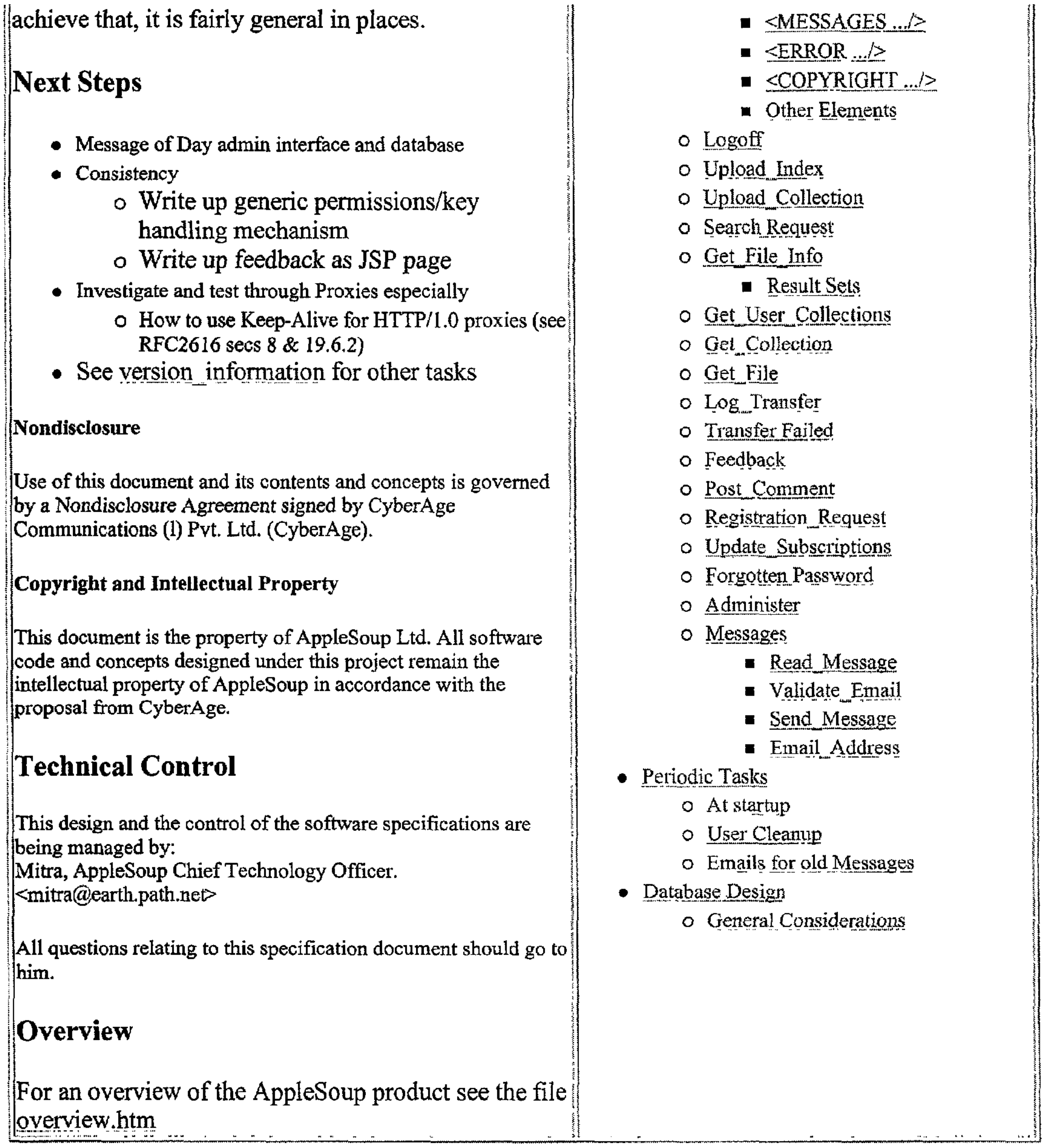
The alpha version of AppleSoup is intended to have the functionality of the version 1.0 release, as presently conceived, to the greatest extent possible. Achievement of the design objective will permit a rapid move to the launch of AppleSoup in mid-February.

The purpose of this document is to provide a design specification for the server portion of the alpha release of AppleSoup. This specification is to be used by AppleSoup developers as their guide in the preparation of the AppleSoup server and client components. The alpha version of AppleSoup is intended to have the functionality of the version 1.0 release, as presently conceived, to the greatest extent possible. Achievement of the design objective will permit a rapid move to the launch of AppleSoup as early as possible.

It is the intention of this specification to define a server that will enable enhancement of the product without significantly rewriting the server code, and to

Back to Spec Index

Contents



achieve that, it is fairly general in places.

Next Steps

- Message of Day admin interface and database
- Consistency
    - Write up generic permissions/key handling mechanism
    - Write up feedback as JSP page
- Investigate and test through Proxies especially
    - How to use Keep-Alive for HTTP/1.0 proxies (see RFC2616 secs 8 & 19.6.2)
- See version_information for other tasks

Nondisclosure

Use of this document and its contents and concepts is governed by a Nondisclosure Agreement signed by CyberAge Communications (I) Pvt. Ltd. (CyberAge).

Copyright and Intellectual Property

This document is the property of AppleSoup Ltd. All software code and concepts designed under this project remain the intellectual property of AppleSoup in accordance with the proposal from CyberAge.

Technical Control

This design and the control of the software specifications are being managed by:
Mitra, AppleSoup Chief Technology Officer.
<mitra@earth.path.net>

All questions relating to this specification document should go to him.

Overview

For an overview of the AppleSoup product see the file overview.htm

Constraints

Time: Due to the fast track process for developing this product, this alpha release needs to be ready for initial review on 28 January 2000.

In order to produce the Alpha as speedily as possible, it may be advisable for some parts of the protocol to only be implemented in the simplest form required to complete the Alpha, and CyberAge are encourage to discuss with Apple Soup any parts of this code they would prefer to delay writing in order to achieve a speedy Alpha.

If any other particular aspect of this specification creates difficulties with meeting that schedule, CyberAge should communicate with AppleSoup immediately so that the correct tradeoffs are made.

Server Architecture

The server architecture will consist of:

- one or more servers running:
    - Linux
    - Apple Soup Server application written in Java
- a server running:
    - Linux
    - MySQL - this may at some point be replaced by a higher end database such as Oracle

During the initial testing these two servers may be physically one box, but this should not be assumed since splitting the middle-layer is critical to scaleability.

Design Considerations

- scalability: The AppleSoup application will originally reside on one server (and the alpha version of the software can be built with that capability). However, it is expected that the AppleSoup server application will eventually reside on many different servers possibly in many different physical locations. This will require that an AppleSoup connection request will be arbitrarily assigned to some one of those servers and will then be executed from that server, as the result of some, yet to be designed, load and location balancing algorithm. The code that is developed should facilitate this process. The actual design of such a network is the subject of another design document which is being developed over the next few weeks.
- simple, reuseable code. The growth of the AppleSoup product is expected to be substantial and rapid., both in terms of volume of users and in terms of functionality of the product. It is imperative that simple, modular coding techniques be used and that all code be constructed using reusable code design techniques. It is imperative that the coding team members be in close communication to ensure that this happens.
- efficiency. The server must be designed to handle a large load, for this reason the code should utilize a design optimized for high volumes of simple transactions rather than low-volumes of complex transactions. Some specific requirements are:
    - The Server should be memory resident, monitoring the port, it should not be launched each time an incoming request arrives.
    - It should, if possible, use threads so that a single process can handle the entire server
    - If threading is not possible, then the Server needs to fork a process pool and use that as needed.

Mitra has extensive experience in designing efficient servers and will be available to help with this process.

- database independent. The initial database for the AppleSoup application is MySQL. It may prove necessary at a later date to migrate to a more traditional database such as Oracle or Sybase so as to have the necessary database engine characteristics for a large volume application. To facilitate an easy transition all database access should be done through functions that can be easily replaced to accommodate a different database engine.
- All interaction with the server, for maintenance activities, should be designed to work in either unix command-line mode, or through a HTTPS/HTML interface on the same, or a seperate port. See the "Administer" command below

Media file Hash function

There is a problem in identifying media files, so that the same file, if resident on different client machines, is so identified. Files are identified throughout the system by a hash function. The function used is MD5 performed on the entire file. This is converted to a 22 character Base 64 string.

Encryption

It is intended that all messages passed between the AppleSoup clients and servers should be encrypted. In the alpha version of the software, this encryption will not be done,in the future it is likely that HTTPS will be used.

Language

Anywhere that the server is presenting a string to the user, it should look the string up from the "languages" table, based on the language presented by the user in the "Accept-Language" http header, if this header is not present then "en" should be assumed. Where a specific Language is not available, then the closest should be presented, so for example if "en-US" is specified and not available then "en" should be used. If none of the specified languages are available then "en" should be used. See RFC2616§14.4 for the format of Accept-Language.

Communications Protocol

The server will communicate with the clients using HTTP or in the future HTTPS over TCP/IP.

The developers are encouraged to use standard HTTP Java classes for this to facilitate future upgrades, and to enable easier support of clients accessing through proxies for example. It is crucial that both client and server ignore any HTTP headers it doesn't recognize as these may have been added by Proxies etc., or have been added at the Server to allow it to work in other circumstances.

The Protocol has been designed to be as state-less as possible, the only state that the server should need to maintain about a connection / user is whether they are connected or not, and the list of files they have available.

This protocol may be changed at some point in the future, so - without performing unneccessary work

- the HTTP specific parts should as far as possible be in seperate portions of the code (classes?) to facilitate a potential change in the future.

Error Messages

The Server returns HTTP status codes as close as possible to the standard ones. The General meaning of the status codes that are likely to be used are:

Code	Meaning	Commands returning
200 OK	Success - results follow	
201 Created	The requested item was created on the server (note HTTP requires the URL of the result in the Location field, we skip that)	
204 No Content	Success - this is used where there is no result to send back.	
400 Bad Request	The syntax of the request was invalid (e.g. two "?") or a hash that contains something other than 22 characters.	
401 Unauthorized	The client needs to Authenticate itself	
403 Forbidden	The server will not allow anyone to do what was just asked	
404 Not Found	The requested item was not found	
500 Internal Server Error	Any internal server error - obviously shouldn't be happening!	
503 Service Unavailable	Returned if the there is a temporary problem, for example unable to open connection to the Database Server.	

Depending on the Server design, some of these might be generated in common code, for example by the HTTP libraries, or for example, a piece of code that parsed tha GET request into a data structure of parameters might return a 400 on invalid syntax.

Refer to RFC2616 for other HTTP error codes, some of which might be generated by libraries or Server code, for example if a client requested HTTP/2.0 the server might return a 505 error code.

In HTTP the Server is encouraged to return an entity (i.e. a document) describing in more detail the error. The AppleSoup Server should be configurable to run in two modes, in Operational mode, the minimal, least helpful error is returned with just enough information so that the client can work properly. In Debug mode, as much information as is available should be returned with the error, to facilitate debugging.

In the command by command descriptions below, only errors specific to that command are described.

Proxies

Proxies can lead to a number of complications, firstly the client has to identify (from the Browser's Internet preferences) that a Proxy is being used.

- The HTTP version number may get mangled, so the server shouldn't be fussy about seeing HTTP/1.1
- The User-Agent should not be mangled, but if we find that it is, then we'll need to include it in the URL.
- The Proxy may refuse to keep an open connection, in this case the server will have to deal with a new connection for each request, this will lead to complications with the client acting as a server of files. *These complications are to be addressed later.*
- The Proxy may require extra headers from the server, and serve up extra headers to the Client, the Client should ignore all headers it doesn't understand
- The Proxy may implement HTTP/1.0 which the server handles a little different using "Keep-Alive".

The Server Application

The server application will be used to handle all areas of interface with client users. It is a modular application which analyses an incoming service request to determine the service needed and then processes that request. This section details the service requests of the application.

Connection Establishment and Authentication and Loss

Authentication background.

Two forms of Authentication are defined for HTTP in RFC2617 Basic and Digest. Basic is too simple and sends passwords in plaintext, Digest is complicated and still wouldn't allow us to check the client is genuine. However, the system was designed to be extended, for example Proxies are required to pass through the authentication info untouched so designing our own became the best option.

In order to check that it is a genuine AppleSoup client, the best thing is to be able to perform a calculation based on the code itself, then a counterfeit client would also need to include current AppleSoup binaries as well.

Connection opening

The Server will receive an incoming HTTP request, and open a persistent connection. In HTTP/1.0 this has to be explicitly stated through "Keep-Alive", in HTTP/1.1 it is the default. Note that although the Clients will all be HTTP/1.1, the Server is likely to have to talk to Proxies running HTTP/1.0.

Unauthenticated

If this request does not contain any Authentication Information (as is the case for the first attempt by the client to talk to the server) and if this command needs authentication (most do, currently only "read_message" does not), then the server will respond with an error code of "401 Unauthorised" and with an HTTP header of:

```
WWW-Authenticate: AppleSoup realm="applesoup.com",
nonce="987654321:1a2b3c4d5e6f7g8h9i0j1k"
```

Where the value of the nonce is defined as:

```
TimeStamp ':' MD5(TimeStamp ':' PrivateKey)
```

where the TimeStamp is the Unix timestamp (number of seconds since 0:0:0 UTC 1970) and the PrivateKey is a random string known only to the server, and should be generated at Server initialization.

Mitra:ToDo: either fix this string over time, or change Read_Message not to use PrivateKey

CodeReview: ToDo: Check this string is random, not fixed as in first version

Authenticated

If the request contains an HTTP "Authentication" header it should take the form.

```
Authentication: AppleSoup realm="applesoup.com", nonce="987654321:1a2b3c4d5e6f7g8h9i0j1k", user="jsmith"
    response="1a2b3c4d5e6f7g8h9i0j1k"
```

The server should check the response using the following algorithm.

First check that the nonce is current, by comparing the TimeStamp (the part before the ':') with the current time, this means that it is less than 1 hour old (that time difference should be configurable). If the nonce is old, then the request should be treated as if it did not contain any Authentication, and a 401 error generated as above with a newly calculated nonce, BUT the string ',stale="1"' should be appended.

If the nonce is current, then check if the users DB contains a cached "authenticate" field, and if so check that whether the entire Authenticate line matches this cached value (a standard string comparisom should be sufficient). If it matches go to "Got-It" below.

If the nonce is current, but there is no cached entry, then check whether the nonce is valid, by calculating MD5(TimeStamp ':' PrivateKey) and checking against the second part of the nonce supplied by the Client. (Note use the TimeStamp from the nonce, not a current TimeStamp). If the nonce is invalid, then a 401 error should be generated as above, and the attempt should be logged as it might be a symptom of a hack.

If the nonce is valid, then check the response matches:

```
MD5( MD5(user ":" realm ":" password) : nonce : exedata)
```

Where "MD5(user ':' realm ':' password)" is stored in the password field of the "users" table in the DB.

Where "exedata" is a portion of the .exe selected by looking at the last few bits of the nonce.

First, the "User-Agent" HTTP header is parsed to discover which version of the Client is being used.

Then a 256 byte chunk is chosen from the server's copy of this Client. This is done by looking at the last two characters of the nonce, **Ashish ToDo: Define algorithm to go from last two chars to bytes from database.** To counterfeit a client would require guessing this algorithm - extremely hard just from packet sniffing, since we MD5 the result, *and* a complete copy of the real client available at run-time.

If the "response" does not match as above then a 401 error is returned with a new nonce.

If it matches, store the Authentication information in users:authenticate, and store the thread information in users:thread and users:server.

Got It!

If this is the first time that Authentication has succeeded on this TCP/IP connection,then:

- If users:online == 1 then signal other server threads that the user needs to be disconnected.
- Store the entire "Authentication" header in the authenticate field of the users table
- Set the "users:location" field to the hex value of the users IP address.

Connection maintenance and Timeout

The client will maintain a connection open, using standard HTTP/1.1 mechanisms. The server should time out idle connections after a configurable period of time, close the TCP/IP connection and treat as a "Connection Loss".

Note also that the Authentication algorithm may time out the "nonce" and send a "401Unauthorised" message in response to a genuine request, in this case, the WWW-Authenticate line will contain 'stale="1",' the client should spot this, and recalculate the Authentication string based on the new "nonce", without re-requesting userid and password

Connection Loss

When a connection with the client is lost, it is possible that either: the user chose to disconnect; or some part of the communications chain broke. In the latter case, the client will attempt to reconnect.

The server has already cached the Authenticate information, in the "users" table, and it should not delete the list of files on the client. If a client then attempts to connect using the cached Authenticate field, then the server should reconnect the client as if nothing had happened.

The server should set the IP Address for the user in the "users" table to 0, and set decrement online (we used to set it to 0, but decrement handles the case of dual login) and set time_left to now.

After a configurable delay the Server should remove the user's file from its list of available files, and delete the cached Authenticate information from the users table. See Periodic:user cleanup.

Information presented at any time

Each time the Server responds, it has the opportunity to include some extra XML into the response. The client should watch for this information, which might have nothing to do with the information asked for, and take action accordingly.

Some specific examples are documented in the Login section which describes how a XML component may be generated during a session for: <SOFTWAREUPDATE.../> <MESSAGEOFDAY ..../> and <MESSAGES .../>

In cases where the server would have responded with a 204 (success but nothing to send back), then it should reply with a 200 status code and the extra entity.

Login

"GET /login?port=1234" should be the first call on every new connection, but not neccessarily if the client is just re-connecting after a communications interuption. The server will store the port in the "users:port" field.

The server will be constructing a XML document to send back to the client which will have zero or more components defined as below. See Comms Protocol: Connection Establishment for the structure of the full document.

The server will set time_left to 0.

<SOFTWAREUPATE .../>

DO NOT IMPLEMENT THIS SECTION YET, THIS WILL CHANGE BASED ON ACTIVE INSTALL

The server should look at the User-Agent HTTP field which will take the form.

User-Agent: AppleSoup/1.2.3456 PC

The server should first check against the cached current build of the client, and if the User-Agent matches, it can omit the SOFTWAREUPDATE element.

The server looks up the User-Agent in the "useragent" table, which will contain information for prior versions of the client. The server should progressively look up the full version and build (major=1 minor=2 build=3456 platform="PC"), if this is not found, it can look up (major=1 minor=2) and if this fails it can look up (major=1). The results include status, the URL of the upgrade value and a textual reason, which should be returned to the client as for example:

<SOFTWAREUPDATE version="1.2.1457" platform="PC" url="http://www.applesoup.com/downloads/upgrade1.2.x.exe" expiry=9876543 why="Fixes a few cosmetic bugs"/>

"expiry" specifies three possible states.

- *If expiry == 0, then the version is still supported.*
- *If expiry == -1, then the version has expired and the server should disconnect this connection after sending the response.*
- *Otherwise, the version is supported but will not be after the date and time specified. The client could display this to the user in the form "This version expires in 10 days, would you like to download and install a new version".*

This procedure allows AppleSoup to decide which prior versions of the client to support, and to provide quick upgrade packages rather than requiring a full download each time.

This XML component could also be generated at any time that the useragent table is changed in a way that effects a logged-in user. This is required since some users with permanent net connections might leave AppleSoup permanently on.

END OF SECTION NOT TO IMPLEMENT

<MESSAGEOFTHEDAY .../> (from Beta:1)

The Server should look for a MessageOfTheDay value in the languages table at stringid=. If present, the XML document should contain a message such as.

<MESSAGEOFTHEDAY text="Welcome back"></MESSAGEOFTHEDAY>

This XML component should also be generated for all existing users when the Message changes during their session. Note that this gives us a requirement for some synchronisation between threads on the server so that one thread knows when this has changed.

<SUMMARY .../>

The Server constructs an XML file giving a digest of what it knows about the client, to allow the client to update any changes.

```
<SUMMARY>
<COLLECTION name="India" description="My photos of India" private="0" hash="1a2b3c4d5e6f7g8h9i0j1k"/>
<COLLECTION name="Pets" description="My furry critters" private="0" hash="1a2b3c4d5e6f7g8h9i0j1k"/>
</SUMMARY>
```

The hash records what is known about the files in the collection. On receipt of this information, the Client can compare this information with its own calculations of these values, and then send Upload_Collection commands for changed collections.

<MESSAGES .../>

The Server should check the "messagelist" table for any messages waiting for this user. If there are any then the XML document should contain:

```
<MESSAGES>
<MESSAGE from="patel" to="jsmith,fred@somewhere.com" subject="Wow" text="this reminds me of home" hash="a1b2
</MESSAGES>
```

The <MESSAGES> component may also be generated during a session when a message arrives for a logged in user.

<ERROR .../>

When the Server generates an error, it should - depending on whether it is running in a debug mode, pass this message back to the client as part of the response. The client should display this in a dialogue box.

```
<ERROR class="Collection" text="Collection name too long"/> "
```

<COPYRIGHT .../>

When the Server sees that a file is copyright (in **TBD)** or illegal, it should pass this message back to the client as part of the response.

```
<COPYRIGHT/>
<FILE hash="1a2b3c" reason="1">
</COPYRIGHT>
```

The client should display this in a dialogue box prompting for deletion.

Other Elements

It should be assumed that other elements will be added to the Login response in the future.

Disconnect

The server may receive a Disconnect command as a simple GET when the Client knows that it is disconnecting, for example because the user has explicitly done so, or shutdown the machine etc. This will be a simple GET

```
GET /disconnect
Host: server.applesoup.com
Accept-Lanuage: en, fr;q=0.5
```

The server should take the actions that would happen had a connection been lost, but can also take the User Cleanup actions that would be applied later.

The server responds with a 204 status code.

Upload_Index

The server should receive a POST containing an XML file of the hash value for each file available from the user's machine.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE AppleSoup SYSTEM "http://www.applesoup.com/support/applesoup.dtd">
<FILE_INDEX>
<FILE hash="12345678901234567890l2"/>
<FILE hash="56789012123456789O1234"/>
</FILEINDEX>
```

The server should parse this file, and add each Hash along with the userid (from the Authentication information) to the "userfiles" table. Note that any previous files that the user has should have been deleted when the previous connection was dropped, but the server should check and delete anyway.

If there are any files in the list with copyright >1 then a <COPYRIGHT....> message should also be returned, files with a copyright >0 should not be added to the database, the operation should still complete successfully.

A success code of 204 should be returned.

Upload_Collection

The server should receive a POST contain an XML file with information about each file in the collection.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AppleSoup SYSTEM "http://www.applesoup.com/support/applesoup.dtd">
<COLLECTIONS>
<COLLECTION name="India" description="My photos of India" private="0" oldhash="la2b3c4d" hash="5e6f7g8h9i0j1
<FILE hash="12345677812123323214 32" type="image/GIF" adult="0" description="Taj Mahal" creator="J.Smith"
file_size="23023" />
<FILE hash="21223342142343242314 23" type="video/AVI" adult="0" description="Elephants at the Taj" creator="F
file_size="1234567" oldhash="12345abcde"/>
</COLLECTION>
</COLLECTIONS>
```

The server should look for a record with matching userid and Collection Name, if the record exists it should be updated, otherwise a new record is created and its fields set.

If the client has included a "hash" attribute of just "" then the Collection has been deleted. Delete from the Collection table and all the relevant elements from "elements" table.

If the Client has included a "oldhash" attribute in the COLLECTION element, then this should be checked against the "hash" field of the "collection" table.

- If it doesn't match, the server should respond with an HTTP 404 error, and the client should resend a complete list with no "oldhash" value.
- If no "oldhash" attribute was supplied then this is a replacement, the server should query the collelements table, delete any files that are no longer in the collection. Then, add any new ones - and set their timestamp to now, and update the information (but not the timestamp) on any files that remain in the collection.
- If a matching "oldhash" attribute was supplied, then this is an incremental update, then for each file supplied:
  - o If there is a oldhash but hash="" then it is a delete, delete the element.
  - o If there is a oldhash and hash value its a change, update the information, but not the timestamp. Note that oldhash=hash is valid and means that the description was changed but the file not changed.
  - o if there is a hash but no oldhash then add the file and set the timestamp

The Server should then set the hash field in the collections table to the "hash" value.

The debugging version of the server can generate and check a new collection hash which is the XOR of the MD5 of the concatenation of the values from the fields sent with each file in a Collection. So algorithmically

- Extract all the items related to the Collection from the COLLELEMENTS table
- Extract the fields hash, type,adult,description,creator,file_size
- Concatenate these fields and do an MD5 on them.
- Starting with a hash of all 0 bits, XOR the MD5 (bitwise, not character wise!) from each of the items.
- Store the result in the Hash field of the Collection table.

Note, that if the operation was an incremental update, then the server can also check the Hash generation, by only producing the hash on the new files, and XOR-ing it into the oldhash and checking against hash.

The collections:size field should be maintained by this operation, i.e. incremented for every element added, and decremented for every element removed.

If the message includes any files with copyright >1 - other than deleting them from a collection - then a <COPYRIGHT/> message should be generated and the files not added to the table or into the MD5, but the operation should still succeed. Files with copyright = 1 may be added to a collection.

Otherwise, the server responds with a success code of 204.

<SUBSCRIPTIONS .../>

Subscriptions will not be implemented till Beta #1

The server should check the "livesubscriptions" table for users who are online currently and subscribed to this collection, for each of them, an XML fragment can be generated.

```
<SUBSCRIPTIONS>
<COLLECTION name="India" description="My photos of India" private="0" oldhash="1a2b3c4d5" hash="e6f7g8h9i0j1
<FILE hash="1234567781212332321432" type="image/GIF" adult="0" description="Taj Mahal" creator="J.Smith" fil
<FILE hash="2122334214234324231423" type="video/AVI" adult="0" description="Elephants at the Taj" creator="F
</COLLECTION>
</SUBSCRIPTIONS>
```

This can be added to the next result returned to the client. Note that the "COLLECTION" element of this should be the same for almost every user, and match the version received with Upload_Collection, since all users should already have the latest version. The exception will be users who have not received any data since the previous update, so an array of recent updates to send might be required. How this is passed efficiently from the thread receiving the collection to those handling the users who need to be informed is left to the server developers.

Search_Request

The server receives a GET containing paramaters for the search,

```
GET /search_request?search_start=0&title=Taj&result_set=&type=image&online=1&adult=1
Host: server.applesoup.com
Accept-Lanuage: en, fr;q=0.5
```

The server should search the "elements" table, the exact match between the query and the search is to be defined later, for now, a title of "Taj" should be matched against "Taj" appearing anywhere in the title field.

Any file with copyright >0 should not be returned.

If type is specified, one or multiple times,then only files with that type or type-prefix are returned, type=club is used for file clubs. For example "type=image" would mean just to return images, "type=club&type=video" would return videos and clubs but not images. If type is not specified then everything is returned.

If online=1 is specified then only files which are online are returned.

If adult=1 is specified then only files with adult =1 then only files with adult <=1 should be returned (i.e. those not specified as adult).

The search_start identified the number (starting with 0) of the first record to send in the result. Since the client is likely to send subsequent requests for the remaining pages of results, the server should cache the results of the search. Note that SEARCH_COUNT in the result is therefore one more than the index of the last search result. The number of results returned in each response should be a configurable parameter.

The result_set field contains the information to be returned, see Get_File_Info:Result_set=1 for its interpretation.

The server should construct an XML document to return of the form, and return it with a success status code of 200.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AppleSoup SYSTEM "http://www.applesoup.com/support/applesoup.dtd">
<SEARCH_RESULTS search_start="0" search_end="50" search_count="75">
<FILE hash="12345677812123323321432" name="taj.gif" type="image/GIF" adult="0" creator="jsmith"
description="Taj Mahal"
file_size="23023"/ >
<FILE hash="21223342142343242331423" name="elephants.avi" type="video/AVI" adult="0" creator="mpatel"
description="Elephants at the Taj" file_size="1234567"/>
</SEARCH_RESULTS>
```

If there are no results to the Search, then an empty SEARCH_RESULTS is returned, with search_start = search_end = search_count = 0.

Get_File_Info

The server will receive a simple GET.

```
GET /get_file_info?
```

```
hash=12345677812123323214328&hash=1a2b3c4d5e6f7g8h9i0j1k&collection=123&comment_start=10
&result_set=2
Host: server.applesoup.com
Accept-Language: en, fr;q=0.5
```

The server should look up the hash in the "files" table, If the result set includes comments, then it should also look up comments from the "comments" table. If the hash is not found, then a 404 status is returned. If the hash is found, then the server should return a 200 status, and an XML file depending on the result_set.

The client can indicate which set of results are wanted, these will be documented below as they are defined.

If the result_set includes comments and the comment_start paramater is present, it specifies where in the list of comments to start sending results.

Any file with copyright >0 should not have any locations returned in the <FILES> section but should have a copyright="x" attribute.

Result Sets

Result Set = 1

All information from the Files table is returned, typically this occurs in a Search or Get_Collection.

```
Content-Type: text/xml
Content-Length: 123

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AppleSoup SYSTEM "http://www.applesoup.com/support/applesoup.dtd">
<FILES>
<FILE hash="12345678901234567890l2"  type="image/GIF" adult="0"
  description="Taj Mahal" creator="J.Smith" file_size="23023">
<FILE hash="1a2b3c"  type="image/GIF" adult="0"   description="Elephants" creator="mpatel" file_size="123456">
</FILE>
</FILES>
```

Result_Set=2

Returns just comments and location. It is significant that it does NOT return descriptions, since these may vary for the appearance of a file in different collections.

Comments are presented according to an algorithm that might be changed later:

- First comments from the collection (as specified in the call) are listed in reverse DATETIME order (newest first)
- Then comments not in any collection in DATETIME order
- Then comments from other collections. in DATETIME order
- Only the first 10 are returned.

The server returns Location information neccessary for the client to select a source, and do a Get_File. Note that this is separated from the search results so that it can be done as late as possible, to maximize the chance that the chosen location is still available. The Server should look in the "userfiles" table for this hash and then do a join between the "userid" field of this table, and the "users" table to locate the remaining fields.If there are no locations online then an empty

<LOCATIONS> element is returned, with no <LOCATION> elements, *at some point we will notify users when a file requested becomes available.*

```
Content-Type: text/xml
Content-Length: 123

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AppleSoup SYSTEM "http://www.applesoup.com/support/applesoup.dtd">
<FILES>
<FILE hash="12345678901234567890l2">
<COMMENTS>
<COMMENT userid="jsmith" datetime="98765432" text="I  like this shot, taken at sunset"/>
<COMMENT userid="mpatel" datetime="98765432" text="I  wish he'd learn to focus"/>
</COMMENTS>
<LOCATIONS>
<LOCATION userid="jsmith" ip="12.34.56.67" port="5556" bandwidth="56000">
<LOCATION userid="mpatel" ip="23.45.67.80" port="5556" bandwidth="26000">
</LOCATIONS
</FILE>
</FILES>
```

Get_User_Collections

The server receives a simple GET.

```
GET /get_user_collections?userid=jsmith
Host: server.applesoup.com
Accept-Lanuage: en, fr;q=0.5
```

It should look up the "userid" in the "usercollections" table (ignoring any records with "private == 1"), and construct an XML file as follows, which should be returned with an HTTP success status of 200.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AppleSoup SYSTEM "http://www.applesoup.com/support/applesoup.dtd">
<COLLECTIONS>
<COLLECTION userid="jsmith" name="India" size="2" description="My pictures of India"/>
<COLLECTON userid="jsmith' name="Thailand" size="4" description="My pictures of Thailand"/>
</COLLECTIONS>
```

If the userid is not found, then a 404 is returned. If the user has no collections then an empty COLLECTIONS element is returned.

Get_Collection

can use a Simple Get, such as:

```
GET /get_collection?userid=j.smith&name=india&result_set=1
Host: server.applesoup.com
Accept-Language: en, fr;q=0.5
```

The server should look up "userid" and "collection_name" in "collections" and "collelements" and construct an XML file of the results which should be returned with an HTTP success code of 200.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AppleSoup SYSTEM "http://www.applesoup.com/support/applesoup.dtd">
<COLLECTION userid="jsmith" name="India" size="2">
<FILE hash="12345677812123323 21432" type="image/GIF" adult="0" description="Taj Mahal" file_size="23023"/ >
<FILE hash="21223342142343242 31423" type="video/AVI" adult="0" description="Elephants at the Taj" file_size="1234567"/>
</COLLECTION>
```

Note that "result_set" has the same meaning as for Get_File_Info and Search_Request.

No files should be included who have copyright >0.

If the Collection is not recognized then a 404 is returned, if there are no files in the collection then an empty COLLECTION document is returned.

Get_File

This is the tricky Client<->Client call, and will in the future be modified to work around firewalls etc. The server has to handle this when the client is requesting a file associated with a Message.

This command is used to allow a client to obtain a file from another client. The Get command passes the file requested by passing the hash of the file identifier. The server queries the database for all known locations of the file on clients that are registered with AppleSoup. This, in effect, creates a massive database of available files which is distributed across all the clients of the AppleSoup network.

If the file is located at a connected location, AppleSoup automatically determines which location should be presented to the client to establish a file transfer. This automatic selection is based on various parameters such as number of files currently being transferred from the donating site, what bandwidth exists on the both clients, etc. In the most general cases the transfer algorithm may select more than one donating site with portions of the file coming from each of the donating sites.

If the transfer is disconnected by the donating client going off-line the client issues automatically switches to another potential donating client so as to continue (without starting anew) the transfer from that location.

It uses a simple GET.

```
GET /get_file?hash=1234567781212332321432
Accept-Lanuage: en, fr;q=0.5
User-Agent: AppleSoup/1.2.3456 PC
```

The result will be a standard HTTP response containing the file.

If the file is not available, or if the file is marked as copyright >0 whether or not deleted, or there is some other problem then a HTTP error code 404 should be returned to the client.

Note that this call when done client <-> client is not Authenticated. At some point, this may be changed.

Log_Transfer

This is sent after the client has successfully complete a Client <-> Client Get_Info.

The server receives a Simple GET

The server should store this along with the timestamp in the "transferlog" table.

The server should return an HTTP status code of 204.

Transfer_Failed

This is sent after the client has failed to complete a Client <-> Client Get_Info.

The server receives a Simple GET

```
GET /transfer_failed?hash=12345677812123323211432&ip=abcdef01&port=1234&reason=tcp_open_failed
Host: server.applesoup.com
Accept-Lanuage: en, fr;q=0.5
```

The server should store this along with the timestamp in the "transferfailure" table.

Other actions may be required as experience is gained, for example we may decide not to send out this IP address if a number of tcp_open_fails are seen.

Note that the ip address is sent in hex, and port in decimal

For privacy reasons, in the future we will probably want to take action and then only keep statistical information, not the hash AND ip.

Santosh:TODO: Supply list of possible error codes THEN tables to be built

The server should return an HTTP status code of 204.

Feedback

Do not implement this, we will use a standard HTML form and handle the results with PHP or JSP.

This is sent by the client to provide suggestions - it is possible that this call will not be used, and instead the client will launch the browser pointed at an HTML form.

This uses a Simple POST:

```
POST /feedback
Host: server.applesoup.com
Accept-Lanugage: en, fr;q=0.5
Content-Length: 54
Content-Type: application/x-www-form-urlencoded

text=Wow,%20I%20really%20like%20your%20service%0a%0dWhen%20is%20the%20next%20release%20due%20out.
```

Note the URL encoding of the Spaces, and Carriage-Return Line-Feed inside the text

The server should store this in the "feedback" table, and return an HTTP status code of 204

Post_Comment

When a member wishes to post a comment on a media file, the client sends a message with the following information:

The server receives a simple POST:

```
POST /post_comment
Host: server.applesoup.com
Accept-Lanugage: en, fr;q=0.5
Content-Length: 54
Content-Type: application/x-www-form-urlencoded

hash=12345677812123323321432&text=I%20like%20this%20shot,%20taken%20at%20sunset&userid=jsmith&name=Ind
```

Note that useid refers to the owner of the collection, the userid of the poster comes from the Authentication information.

If the file being commented on has copyright > 1then the comment should not be added and a <COPYRIGHT> element generated.

The server should append this information to the "comments" table.

The responds with an HTTP status code of 204.

Registration_Request

When AppleSoup is first installed, the new member undertakes a registration process. The is handled by the following message:

The server receives a Simple POST:

```
POST /post_comment
Host: server.applesoup.com
Accept-Language: en, fr;q=0.5
Content-Length: 54
Content-Type: application/x-www-form-urlencoded

userid=jsmith&password=a1b2c3d4&email=jsmith@nowhere.come&notice=12&connection_speed=56k
```

The server should look up the userid in the "users" table.

The password is created by the client from the password entered by the user, MD5(userid ':applesoup.com:' password). This means that the plain-text password is never sent across the net which is important because its probably used by the user on other web sites. This is the only time that the hashed value is sent over teh net, and ideally this call should be hidden in the future by HTTPS.

If the userid is not in the table, then the server should add the record, and add the password hash, and return a status code of 204.

If the userid is in the table, AND this message is authenticated to come from that user, then the registration information and hashed password is updated, and a status code of 204 returned. Note that it is not possible for this request to change the userid.

If the userid is in the table, but this message is not authenticated from that user, then a status code of 400 should be returned with a string "NAME_FAIL".

Update_Subscriptions

Subscriptions will not be implemented until Beta-1

After logging on the Client can update its subscriptions. The Server receives an XML post

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AppleSoup SYSTEM "http://www.applesoup.com/support/applesoup.dtd">
<SUMMARY>
<COLLECTION userid="jsmith" name="India" description="My photos of India" private="0" hash="1a2b3c4d5e6f7g8h
<COLLECTION userid="mpatel" name="Pets" description="My furry critters" private="0" hash="1a2b3c4d5e6f7g8h9i
</SUMMARY>
```

this tells the Server the list of Collections that the client has subscribed to, the hash has been generated using an algorithm that matches that in Upload_Collection or it can be that sent by the Server with the last update.

The server generates an XML document with the following algorithm.

- For each collection the user is subscribed to.
    - If the hash != collection:hash
        - For each record for this collection from the "elements" table.
            - It should add the record to a XML fragment it is building that matches that in Get_Collection
            - It should accumulate a hash using the same algorithm as in Upload_Collection
            - If the accumulated hash matches the one specified in the "Summary" then discard the XML fragment, and start afresh to accumulate it based on the remaining records from collelements.
    - Add the (possibly empty) XML fragment to the document being returned, if the hash had matched then include the oldhash in the <COLLECTION> element, always include the hash.

Return a status code of 200, and the XML document which should look like.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AppleSoup SYSTEM "http://www.applesoup.com/support/applesoup.dtd">
<COLLECTION userid="jsmith" name="India" size="2" hash="5o6p7q" description="My photos of India">
<FILE hash="12345677812123323211432" type="image/GIF" adult="0" description="Taj Mahal" file_size="23023"/ >
<FILE hash="21223342142343242311423" type="video/AVI" adult="0" description="Elephants at the Taj" file_size="1234567"/>
</COLLECTION>
<COLLECTION userid="ppatel" name="Taj Mahal" size="2" oldhash="6h7j8k9l" hash="1a2b3c4d" description="A selection of Taj shots">
<FILE hash="21223342142343242311423" type="video/AVI" adult="0" description="Elephants at the Taj" file_size="1234567"/>
</COLLECTION>
<COLLECTION userid="mitra" name="Big Critters" size="2" oldhash="6h7j8k9l" oldhash="3c4d" hash="3c4d" description="Huge animals I've seen">
</COLLECTION>
```

Note the three cases here, the first "India" is where no hash was found, for example a file might have been deleted or its description changed, so a full listing is sent, in the middle case "Taj Mahal", only files were added so an incremental version is sent, in the last case there are no changes.

Forgotten_Password

If a user forgets their password, they can request a reminder. The Server receives an simple GET.

```
GET /forgotten_password?user=jsmith
Host: server.applesoup.com
Accept-Lanuage: en, fr;q=0.5
```

The server should change the password to something new and reasonably memorable, specifically the concatenation of two randomly chosen english words e.g. "palmrisk" this should be sent by email to the main email address of the user (taken from the users table).

This message will be read from two strings in the servers configuration, exchanging \U and \P for the userid and password.

When the user receives the message in their email, they can try again typing in the new password. From Beta 0 they can also go to Tools Menu -> Change Password.

Note that there is no way for the server to determine the original password of the user since the server only stores an encrypted version.

The server returns a success code of 204

It might be simpler to implement this in PHP or JSP so that the actual email message sent can be generated from a template?

Messages

This group of commands handle messages

Validate_Email

This is a browser call, implemented in JSP. The server receives a GET for a URL like:

http://www.applesoup.com/validate_email?userid=jsmith&email=jsmith@foo.com&key=12345

The server should check that the key matches MD5(privatekey ':' userid ':' email)

- If it doesn't match - generate a 403 (or a 404 if that is difficult)
- If it matches generate a page from a template, and set the validationflag on the appropriate field in emailaddresses

Read_Message

This call is generally from a browser, not an AppleSoup client, and so will be un-authenticated, it will be written in JSP on a seperate server.

The Server receives a Simple GET:

```
GET /read_message?id=12345&key=a1b2c3
Host: www.applesoup.com
Accept-Lanuage: en, fr;q=0.5
```

The id refers to the "rowid" in the "messageemail" table.

The key should match the random number created by Send_Mesasge, if it is not then a 403 error is returned (you can't do this). (A 404 error is acceptable if it is not easy to generate 403 errors).

The server will generate a page from a JSP template. The template will presumably contain an <IMG> tag containing the image, which may be served up by any convenient mechanism. The template may contain other URLs including ones for downloading the client, and for blocking spam. Those URLs will be defined later. **Beverly will supply this Template, after inital coding to a dummy template.**

Initially the actual image will initally be served up from a file area shared via NFS between the web servers and the application servers. At some point after some research this will be moved to a more efficient mechanism such as binary objects in MSQL or via a seperate HTTP server with the AppleSoup server posting the file to it, and the IMG tag pointing to it for a GET.

After generating this page, the messageemail item should have the time_received set to the current date & time, and the count in "messages" decremented. See Periodic:Old Messages for what happens when this reaches 0.

Send_Message

When a user wishes to send a file to other people, they fill in a screen.

The server receives a Simple POST:

```
POST /send_message
Host: server.applesoup.com
Accept-Language: en, fr;q=0.5
Content-Length: 54
Content-Type: application/x-www-form-urlencoded

to=jsmith@foo.com&to=mpatel&to=mitra@earth.path.net&subject=Wow&text=Reminds%20me%0aof%20home&hash=a1b2c3&na
```

If the message has copyright >0 then a <COPYRIGHT> element should be generated and returned with a 404.

The server should first check the "to" addresses without taking action on any of them.

- any containing an "@" should be looked up in the "emailaddresses" table
    - any that appear should be replaced by AppleSoup id's.

  - any that look like invalid email addresses are an error
- the rest are presumably AppleSoup id's and should be looked up in the "users" table.
  - *if it is not in the table this is an error*
- In the case of any error, then the message should be abandoned, and HTTP error 404 should be returned followed by an entity containing the XML:

```
<MESSAGE_FAILURE>
<UNKNOWN_RECIPIENT to="mpatel"/>
</MESSAGE_FAILURE
```

on success, the message is added to the "messages" table and the AppleSoup id's or email id, added to the "messagelist" table and any email addresses added to the "messageemail" table, and added to, or updated in, the "viralemail" table. "count" is set to the total number of recipients. Seed is set to a random string.

An email message is generated for each user that is not an AppleSoup user, this should be read from a template file, with simple substitutions of the fields from the message the message will include a URL http://server.applesoup.com/read_message?id=123456&key=a1b2c3 where the id is the rowid from the messageemail table and the key is the random number from the same table. *(Note to developers, keep it simple, don't invent a whole template language).*

See Read_Message for how this is received by an email recipient.

An email is generated for each user that is an AppleSoup user, but has not been connected for some (configurable) period. This should use a different template (which will invite the user to try AppleSoup again).

The Server should look to see if it already has this file stored because of a previous Send_Message, if not, it should issue a Get_File to the client to fetch the file, and store it locally.

See Periodic:Emails for old messages where a process is described for notifying AppleSoup users who don't pickup their messages.

Email_Address

The server receives this when a user wishes to change the list of email addresses supported by AppleSoup. (See Client)

The Server receives an simple GET.

```
GET /email_address?add=jsmith@foo.com&delete=jk@lm.no&pq@rs.tu
Host: server.applesoup.com
Accept-Lanuage: en, fr;q=0.5
```

If their are any add= parameters, then the server should send a verification message generated from a template and containing:

http://www.applesoup.com/validate_email?userid=jsmith&email=jsmith@foo.com&key=12345

where key is MD5(privatekey ':' userid ':' email) to each of them, and add them to the "emailaddresses" table with validationflag=0.

If there are any delete= parameters, then the server should remove them from the list in emailaddresses (provided that the userid matches). If any of the addresses removed matches users:email, then the next validated email address should be made the primary email address. If there are no validated email addresses (the client should not request this) then leave users:email unchanged.

The server should then generate and return an XML file with status code = 200.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AppleSoup SYSTEM "http://www.applesoup.com/support/applesoup.dtd">
<EMAILADDRESSES>
<EMAILADDRESS email="ab@cd.de">
<EMAILADDRESS email="jsmith@foo.com" validationflag="0">
<EMAILADDRESS>
```

Note that this shows which of the addresses have not yet been marked as validated.

Administer

The Administer functionality is used via a normal browser rather than an AppleSoup Client. It is to be implemented in JSP (or PHP) and interact with the main server via the database. Some synchronisation method will be needed between the administrator and the main servers.

Its is important, that however the design, the Server responds quickly to changes, for example if a MessageOfTheDay is changed then it should be sent out in the next response to each client.

The Server should expect something like.

```
GET /administer
Host: server.applesoup.com
Accept-Lanuage: en, fr;q=0.5
```

It is essential that the server validates any administration transactions against a list of administrators.

In response to an authenticated GET as above, the Server should generate an HTML page from a Template. This page should contain:

- Any alerts that administrators need to know about - these will be defined later as we gain operational experience, but might include such things as a disk being filled above a certain percentage.
- Forms for changing things that are frequently changed, in particular:
    - Message of the Day
- Statistical Information extracted from the tables.

The exact specification for this functionality is left vague at the moment, it should be considered as the single place for adding administration functionality. Commands that change things can be defined in whatever way is convenient for the implementers, for example to change the Message of the Day, the client might send another GET

```
GET /administer?messageofday="
    Check%20out%20the%20new%20collections%20from%20award%20winning%20photographer%20Joe%20Brown"
Host: server.applesoup.com
Accept-Language: en, fr;q=0.5
```

Please discuss any functionality being added here with this specifiation's author (Mitra).

Periodic tasks

The server is expected to perform certain periodic tasks on a regular basis. Or as a result of certain conditions.

At startup

At startup, the server needs to perform certain actions - there will certainly be others in addition to this list.

- Check for cleaning up result of this server having previously crashed.
  - Set any users from user table that have this server id as if they had logged off, i.e. set IP address to 0 & set time_left to now. but note do NOT remove the userfiles or livesubscriptions since the user may be in the process of logging back in on another server.
  - Note that a restarting thread should do the same thing but just for users of that thread.

User Cleanup

Users who disconnect do not have their entries removed immediately, in case the disconnection was inadvertant, in which case there will be another connection. After a configurable period (~10mins), users entries with an IP address of 0, and cached Authentication information should have the cached Authentication information removed, and any entries for that user in the "userfiles" or "livesubscriptions" tables removed.

Emails for old messages

When a message is left for an AppleSoup user, but not picked up for a configurable time (~48 hours) then an email should be generated from a template, informing the user that they have waiting messages and prompting them to use AppleSoup to fetch it.

Old Messages

Once a message has been seen by all recipients it should be deleted after a configurable amount of time (1 day). This could be implemented by periodically sweeping through the messages table setting anything with count=0 to count=-1 and deleting any where count=-1.

Any message older than 32 days (configurable) should be deleted.

When a message is deleted, if the file refered to is not refered to by any other message then it should be deleted.

Database Design

General Considerations

All database accesses should be performed through functions that provide a layer which makes the application code independent of the particular database that is used. It is possible that the database might be migrated from MySQL to Sybase or Oracle at a later stage as the database grows in size.

MySQL does not support such concepts as referential integrity, or record locking. It is not intended that we will ever rely on a database engine to provide such functionality. Similarly it is not intended that we will ever rely on triggers or stored procedures for functionality. The design of code and the data dictionary should be done with this in mind while at the same time keeping in mind the need for scalability.

There will be a need for additional tables to support, as yet undefined, statistical data associated with the applications.

The Database table specification is in a seperate document since it is being revised independently of this specification.

AppleSoup
Database Specification

(Company Confidential)

Updated: 4th May, 2000

Purpose

The purpose of this document is to provide a design specification for the database structure as implemented in the AppleSoup server. This specification is to be used by AppleSoup developers as their guide in the preparation of the AppleSoup server and client components. The alpha version of AppleSoup is intended to have the functionality of the version 1.0 release, as presently conceived, to the greatest extent possible. Achievement of the design objective will permit a rapid move to the launch of AppleSoup in mid-February.

It is the intention of this specification to define a protocol that will enable enhancement of the client without significantly rewriting the communications code, and to achieve that, it is fairly general in places.

Nondisclosure

Use of this document and its contents and concepts is governed by a Nondisclosure Agreement signed by CyberAge Communications (I) Pvt. Ltd. (CyberAge).

Copyright and Intellectual Property

This document is the property of AppleSoup Ltd. All software code and concepts designed under this project remain the intellectual property of AppleSoup in accordance with the proposal from CyberAge.

Technical Control

Specification Index

Contents:

This design and the control of the software specifications are being managed by:
Mitra, AppleSoup Chief Technology Officer.
<mitra@earth.path.net>

All questions relating to this specification document should go to him.

Overview

The tables of the AppleSoup database can be considered to occur in three groups.

1. Application Tables: providing the functionality of AppleSoup
2. Log Tables: These accumulate information about what has happened, they can be truncated without damage to the application, although of course this should be done AFTER the information is analysed.
3. Statistical tables: these will be used to monitor the use and growth of AppleSoup. This set of tables might well change a fair bit as we gain experience with the product. Generally these tables will need a various cron type tasks defined so as to populate them. This functionality is not critical to the Alpha development of the product, but needs to be there for launch.

It is critical that anything using these tables should assume that they might change in the future, in particular all code that relates to maximum size of strings should be configured from header files so that the string lengths can afterwards be changed in only a few places (e.g. Database creation routine **where?**, Server: DBClasses.DBConstant and Client C++ config file **where?**)

Application Tables

Abbreviations

Abbreviation	Means	Comment
uint	Unsigned Integer	10 bytes
datetime	BigInt20	Holds a unix timestamp
USERID	CHAR40	Long enough for a AppleSoup userid
EMAIL	CHAR40	Long enough for any reasonable legal email address
HASH	CHAR22	Holds a Base64 encoded MD5 value

IPPORT	CHAR8	IP number and port expressed in hex
FILENAME	CHAR40	File name

The following database tables will be used in the design.

configuration

This is a table used to contain various configuration values used by the software. It has four columns:

Column	**Datatype**	**Description/Comments**
rowid	uint	autoincrement
valuename	CHAR25	the name of the value in this row
value	CHAR80	the value associated with the valuename
description	CHAR64	the description of the valuename

Specific Rows that are needed in this table should be documented here. Please email mitra if you add any rows to this table.

Valuename	**Example Value**	**Description**
DBVERSION	1	Version number of database, increment each time a field is added/deleted to ANY table
SWMINVER	220	Minimum version number of server that this database is compatable with. Note that the server version number and minimum version of database are compiled into the server

size 181 bytes per entry

users

This table contains the list of AppleSoup registered users. An entry is made in this table when a new user registers to use AppleSoup. On subsequent logins and logoffs the online column and location columns are updated. A process needs to run from time to time to update the table for disconnected users who did not logoff.

Column	**Datatype**	**Description/Comments**
rowid	uint	autoincrement
userid	USERID	user's login id
password	HASH	HashedPassword: The value MD5(userid ':' realm ':' password) ; note t the same information as generated by /usr/sbin/htdigest as supplied wit
connection_speed	CHAR10	from initial registration
online	uint	1 if on-line, 0 otherwise
time_created	datetime	Unix timestamp when they registered

location	IPPORT	IP Number and port expressed in hex
authenticate	CHAR150	Cached Authentication response
email	EMAIL	Email address of user
port	uint	port to contact the client on
server	uint	index into servers table of the server handling this user
thread	uint	thread number (or other identifier) on that server.
notice	uint	frequency in days requested for notification
time_left	datetime	Unix timestamp of when disconnected, set to 0 during session or after

Total Size 370 bytes per user

emailaddresses

This table contains the email addresses that a user has. When a user registers, one email address is added to this table. Subsequently (as a 'preferences' capability on the client) the user can add additional e-mail addresses. This requires a suitable message package and a server processing of same. When AppleSoup is installed, a validation code is generated and passed to the client. To receive messages, the member must validate by entering this code. When that is done the validationflag is set to 1

Column	Datatype	Description/Comments
rowid	uint	autoincrement
userid	USERID	user's login id
validationflag	CHAR1	0 when AppleSoup installed, 1 when member has validated their email
email	EMAIL	Email address of user
validationcode	CHAR10	Used during validation process

Total size 101 bytes per email address

userfiles

This table contains the media files that are currently available on the AppleSoup network, identified by users that have them. when you log in your index is uploaded into this table; on log off your entries are purged. If a user attempts to access a file that is no longer connected, an error is generated

Column	Datatype	Description/Comments
rowid	uint	autoincrement
hash	HASH	hash value of the media file
userid	USERID	user name of user having this file
time_entered_online	datetime	connect timestamp
connection_speed	CHAR5	from users file

Total Size 97 bytes per file online

files

This is the table of AppleSoup media files. Whenever a new file hash is loaded in an index, a row is added to this table.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
hash	HASH	hash value of the media file
mimetype	CHAR20	image/jpeg, image/gif, etc.
adult	CHAR1	flag 2 = adult 1 = unmarked, 0 = family friendly
copyright	CHAR1	flag 0 = OK 1 = suspended 2 = copyright 3 = illegal
filesize	uint	in bytes
thumb	HASH	hash value of the media file's thumbnail, if any
notransfers	uint	the number of transfers of the media file
lasttransfer	datetime	timestamp of most recent transfer
private	CHAR1	1 = by invitation, only , 0 = public
creator	USERID	The creator of this file where known
online	uint	count of number of copies of this file online at the moment

Total size = 167 bytes

comments

This table contains comments contributed by viewers on media files.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
hash	HASH	hash value of the media file
timestamp	datetime	timestamp when loaded
comments	CHAR255	the comment entered by the member
userid	USERID	username from login of the commentator
collectionid	uint	optional -- only if in collection

Total size 357

collections

This table is a list of media collections (file clubs), identified by owner. This is one of the tables that is searched when a search query is processed. This table is searched first and if the max number of hits is not achieved, then the elements table is searched

Column	Datatype	Description/Comments
rowid	uint	autoincrement
userid	USERID	username from login
name	CHAR60	file club name
description	CHAR255	description of file club
private	CHAR1	flag 1 = by invitation, only , 0 = public
timestamp	datetime	timestamp when loaded
hash	HASH	A hash of the values sent with the last Upload_Collection
size	uint	Count of elements for this collection
imagecount	uint	Number of images in the collection
videocount	uint	Number of videos in the collection

Total Size 438 per collection

elements

This table contains the descriptions for a media file as added by various file club owners of that image. This is one of the tables that is searched when a search query is processed.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
collectionid	uint	rowid in collection table
filename	FILENAME	name of file as it appears in this collection
time_added	datetime	time at which the fileclub element was added
hash	HASH	hash value of the media file
description	CHAR255	file club owner's description of media file

messages

this table stores messages that are being sent to a friend. the number column is decremented when a message is sent. When it zero the message is deleted. A message is added to this table if it can not be delivered at the time the server processes it. Messages to e-mail addresses will be sent automatically by the server so this only applies to messages to AppleSoup users that are not logged on at the time the message is sent to the server. A purge routine should remove messages after some period of time.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
username	USERID	user's login id of message sender
sendto	CHAR255	list of recipients (applesoup and email) in text form
subject	CHAR80	subject string sent from client
messagestring	CHAR255	message string sent from client
time_added	datetime	time at which the message was added to the table
hash	HASH	hash value of the media file being referenced in the message
number	int	the number of messages remaining to be sent
name	FILENAME	the name of the file
collectionid	uint	optional - if there is an identifyable collection associated with this me

Total size 733 bytes per message

messageemail

Column	Datatype	Description/Comments
rowid	uint	autoincrement
messagenumber	uint	the rowid of the message in the messagetoafriend table
to	EMAIL	email address of this recipient
time_received	datetime	time at which message (not the email message, the web page) read by t initially set to 0
key	CHAR10	initialized to a random value when the record created

Total size 90 bytes

messagelist

the list of addressees for a message. When a user logs on a search is done of this table and those messages that appear are delivered. The corresponding rows on this table are deleted and the counts in the messageforafriend table are decremented. (NB this requires a message structure to exist on the client to store such messages until they are viewed

Column	Datatype	Description/Comments
rowid	uint	autoincrement
messagenumber	uint	the rowid of the message in the messagetoafriend table
userid	USERID	userid of the AppleSoup user that the message is for
reminded	datetime	When a reminder was sent via email (typically 48 hours after msg send

Total size 80 bytes

searchquery

this table logs all queries that are used in searches of the file descriptions. The maximum number that can be returned is a system parameter of the server.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
querystring	CHAR80	query string sent from client -- length to match data transferred
number	uint	the number of results from the query

feedback

This table contains answers from responses to forms - a set of answers from a single form or set of forms is grouped by a response_id. The table is filled by the JSP code behind pages like feedback.htm, question_id comes from the NAME field of a form, and answer is the result (URL encoded). Note that the forms may include hidden fields set by parameters in the URL, such as for example the user's applesoup id.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
time_added	datetime	time at which the transfer took place
response_id	HASH	same for all queries during same session with web server
question_id	CHAR40	id given in <QUESTION ID attribute
answer	CHAR255	answer as entered by the user.

Total size 347 bytes

language

contains multilingual strings for use by the server, the server will look for the "English version of string 10"

Column	Datatype	Description/Comments
rowid	uint	autoincrement
stringid	CHAR20	a message id
language	CHAR20	language of string as specified in rfc1766 which refers to ISO639 an "en" or "en_US"
text	CHAR255	text of string in UTF8

Total size 305 bytes

Specific Strings are:

stringid	**Usage**
1	Message of the Day

viralemail

This table lists each email address that is used in the messaging system. The first time the email address appears a new row with that address is added with the inittime & recenttime time stamps being set, and regtime being null and the dontsendflag being set to 0 and count to 1. Each time a subsequent mail is sent to this same address, the recenttime datetime stamp is set with that time and count incremented. If the address is registered as an AppleSoup user, the regtime timestamp is set. If mail to this address bounces back for invalid address, the dontsendflag is set to 1 and a message sent to the sender. If the recipient sends a don't send message the dontsendflag is set to 1. (note: may want to refine this so that more than one not delivered is required before it is removed.

Mitra:ToDo - add this functionality above to Server spec

Column	**Datatype**	**Description/Comments**
rowid	uint	autoincrement
emailaddress	EMAIL	the email address to which the message is sent
dontsendflag	CHAR1	0 if okay to send mail to this address (default), 1 if recipient declines invalid
inittime	datetime	time at which first email was sent to this address
recenttime	datetime	time at which the most recent email was sent to this address
regtime	datetime	time at which the reciepient registered a copy of AppleSoup
count	uint	incremented each time a message is sent

Total size 121 bytes

livesubscriptions

contains a list of which currently online users are subscribed to which collections.These entries are removed on logoff.

Column	**Datatype**	**Description/Comments**
rowid	uint	autoincrement
userid	USERID	userid of subscriber
owner	USERID	userid of owner of collection (as in collections:userid)
name	CHAR40	name of collection
hash	HASH	hash of last version seen by user.

Total size 152 bytes per subscription

servers

contains any information that needs to be kept across servers. Note that the rowid is used as an index in other tables. This table will be expanded to hold any information that needs to be accessable across different servers, for example to an administration program.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
IP	IPPORT	ip address and port in hex

Log Tables

transferlog

This table is the list of successful transfers. Note that userid is NOT recorded.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
time_added	datetime	time at which the transfer took place
hash	HASH	hash value of the media file
demographics		fields will be added here to record demographics to be specified by cat be obtained from the users table, but userid will not be recorded

Total size 52 bytes

searchlog

This table logs searches. Note that userid is NOT recorded.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
time_added	datetime	time at which the search took place
term	CHAR80	
number_results	uint	number of results returned

Total size 120 bytes

transferfailurelog

This table logs transfer failures. Note that userid is NOT recorded.

Column	Datatype	Description/Comments
rowid	uint	autoincrement
timestamp	datetime	time at which the search took place
hash	HASH	hash of file that was being transfered
ip	IPPORT	IP number and port (from user's table) that was being contacted
reason	CHAR40	reason for failure - this may have internal structure to be defined.

Total size 100 bytes

Statistical Tables

The following tables are used to monitor various aspects of the usage of the database. They are generally populated with snapshot data which are recovered by processes that are spawned by a scheduled (e.g. cron) process.

See inprocess.htm for changes to this

logonsession

a record of the logon sessions

Column	Datatype	Description/Comments
rowid	uint	
userid	USERID	userid
logon timestamp	datetime	the time at which logon occured
logoff timestamp	datetime	time at which the logoff occured

Total size 90 bytes

numberofdownloads

gives the number of downloads to that time. This is a process that goes to the download site(s) and registers the count at those sites at the time

Column	Datatype	Description/Comments
rowid	uint	autoincrement
value	uint	the count
site	CHAR10	the download site
timestamp	datetime	time at which the count was taken

total size 50 bytes

counters

This is a generic table for counting something for statistical purposes.

Column	**Datatype**	**Description/Comments**
rowid	uint	autoincrement
id	CHAR20	an id
count	uint	the count
timestamp	datetime	time at which the count was taken
description	CHAR80	Description of the counter - e.g. who is using it, or what it is counting

Total size 140 bytes

This table is used to record an incrementing, or periodic count. A process wishing to increment the count for id=foo would increment the count for the record with id=foo and timestamp=0 (or null or whatever). Alternatively a process just wishing to record a count can create a new record with a specified id and current timestamp.

Periodically - and this might vary based on id - the server will set the counts, and create new records with timestamp=0.

Consistency Checks

The following consistency checks can be applied periodically to check the integrity of the database.

- All of {livesubscriptions:userid, livesubscriptions:owner, etc} exists in users:userid
- Foreach collection in collections { collection:size = count(elements: elements:collectionid = collection:rowid) }

AppleSoup Client - Specification

(Company Confidential)

Last updated: 20th April, 2000 by Mitra

Purpose

This document is undergoing a UI revision - it should be close enough to being stable for implementation of the new UI to start.

The purpose of this document is to provide a design specification for the client portion of the alpha release of AppleSoup. This specification is to be used by AppleSoup developers as their guide in the preparation of the AppleSoup client component.

The alpha version of AppleSoup is intended to have the *functionality* of the version 1.0 release, as presently conceived, to the greatest extent possible. Achievement of the design objective will permit a rapid move to the launch of AppleSoup.

This is one of several specification documents which detail the design of the AppleSoup application. This document will refer to other documents in this specification series and they should be read in conjunction with this document to obtain a complete view of the current design concept.

The reader of this document should also be familiar with the general overview of the AppleSoup Product.

Note: We intend that we will have beta versions of the AppleSoup client running for Mac Linux clients.

Specification process ToDo and Next steps

Contents

Back to AppleSoup Specifications Index

- Pass through doc and handle **TODO** items
- Put protocol stuff into actions e.g. Get_File_Info etc
  - THEN: Check that all error cases from server are handled.
- Spec items from version_information documen

Nondisclosure

Use of this document and its contents and concepts is governed by a Nondisclosure Agreement signed by CyberAge Communications (I) Pvt. Ltd. (CyberAge).

Copyright and Intellectual Property

This document is the property of AppleSoup Ltd. All software code and concepts designed under this project remain the intellectual property of AppleSoup in accordance with the proposal from CyberAge.

Technical Control

This design and the control of the software specifications are being managed by:
Mitra, AppleSoup Chief Technology Officer.
<mitra@earth.path.net>

All questions relating to this specification document should go to him.

Overview

For an overview of the AppleSoup product see the file overview.htm

TimeLine

This specification is on a fast-track, the tentative dates we are aiming for are:

Well before April 15th	Internal Beta with all "Must-Have" features, source trees are split so that from this point on we always have a stable Beta.
April 15th	Beta-0, with all complete/stable feature sets (timing chosen to be before US students break for summer).
May 15th	Launch of Beta 0
ASAP	Additional Beta versions B1..B4 launched, fixing critical bugs and adding new feature sets

ASAP	Full launch - but marketing to pick date based on competitors, stability and feature sets included.

The set of features for each version are listed in version_information.htm.

If any feature of the specification makes it hard to make this date, then CyberAge are encouraged to contact AppleSoup (Mitra) to discuss changing it.

Design Philosophy

General Design Characteristics:

We are looking for simple, working code which has the following characteristics:

- Easy extensibility; any currently defined functionality is a subset of the anticipated functionality for the first version release.
- Common look and feel; the screen interface should use a common approach to all user interaction.
- A simple interface which does not have a lot of pop-up windows; emulating simple web-style interfaces.
- We are looking for ease of use, from installation through application.
- The code must be easily maintainable.
- button elements, forms, not lots of graphics

Implementation Priorities

An important part of the alpha project is to demonstrate that:

1. Each of the technical concepts involved in the design will work, and
2. When linked together as a product, the various pieces work as a whole

The set of functions to be included in the Alpha version is documented in the Version Information document.

Client Technology Architecture

1. The alpha client application will be written in MS Visual C++, version 6.0. We intend that, to the greatest extent possible, the client should be written using standard features of this tool. Deviations from standard features will be evaluated on a case by case basis, as they appear, by AppleSoup technical staff.
2. The alpha client application will be written to run on the MS Windows 9x operating system. It is intended that the AppleSoup installation process should be a simple as possible and should not adversely impact the functioning of other applications or the operating system. It is anticipated that registry values will be defined and utilized. Any .dll's that are required need to be defined at an early stage and approved by AppleSoup technical staff.
3. We plan to have Mac and Linux versions for the beta version of the AppleSoup client.
4. The ideal target size for the completed, fully functional (version 1.0) product is less than 1MB .EXE installation file, ideally more like 500KB. This alpha version of the client should

have substantially the functionality of the version 1.0 release. We intend to use third party controls for presenting media files as appropriate to speed development time.

5. The connection protocol for the client to the server is defined in the Communications Protocol Document and should be read in conjunction with the Server specification.

Multilingual Capabilities

The AppleSoup application is a tool that can be used by members from a wide variety of languages and cultures. When a potential AppleSoup member visits the AppleSoup website they will be able to select a language of their choice and then be presented with the option to download AppleSoup in that language.

To ensure that this functionality is available, the alpha version of the software will support at least two languages. One of these should be American English. The other can be some second language that is easily available to the developer (e.g., one of the languages, other than English, of India). The text for a Korean version will be supplied to the developer by AppleSoup Ltd.

We should use standard MS VC++ techniques for creating multilingual applications right from the beginning of the alpha development.

To support more than one language, there needs to be appropriate change in the UI in button controls as they should be sized such that the text in different languages can be supported. R&D needs to be done for this, and width of the buttons needs to be changed depending on the text to be displayed on the buttons.

For design purposes, we need to look at different ways of how multilingual support can be achieved. Either there needs to be different exes for different languages, which is a straight forward solution or there needs to be support for having two languages in one exe. How to support multiple languages in one exe and whether it is possible stills needs to be figured out.

It is neccessary that whatever the language chosen for the UI that the client can display messages that are written in another language. For this reason UTF8 will be used for all strings passed through the protocol, and the client should support displaying these strings subject only to the neccessary resources (e.g. Korean fonts) being available on the client.

The design of the UI for the installer will depend on whether there is one installer for multiple languages or one installer per language.

Actual translation in different languages needs to be done for which the translators will be outsourced.

Interface

The section below defines the UI for the Alpha, note that Screen Shots refer to the current version, and so will not neccessarily match the text.

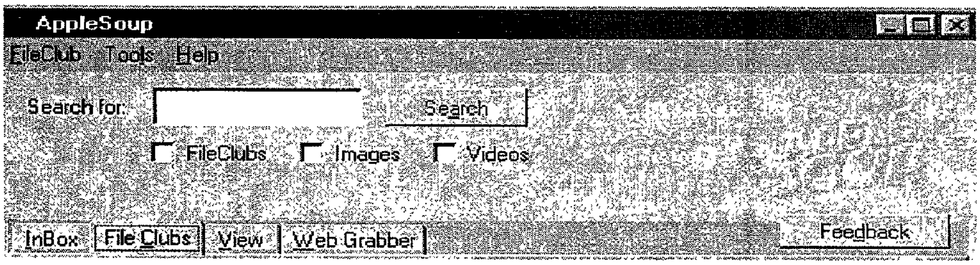
Startup Functionality

After installation of AppleSoup, it will have a small icon in the system tray from where the AppleSoup application can be initiated. The way it works in different modes is described below.

- OFFLINE MODE (this is where AppleSoup starts).
    - If the user has never logged in before, then if the UI is open, then open the Login Panel, do not move out of this mode until a userid and password are available.
    - If "explicit_disconnect" is not set, then when net connection becomes available, go to CONNECTING MODE
    - When user hits connect or selects from systray menu
        - Dial internet connection if this is how the user's dialup connection preferences are set
            - If this fails
                - If UI is open show error message
                - Stay in Offline Mode
            - If it succeeds go to CONNECTING mode
- CONNECTING MODE
    - Connect to Server
        - If this fails
            - If UI is open show error message
            - Wait and try again
        - If it succeeds, Go to ONLINE MODE
- ONLINE MODE
    - Obtain the userid/password from the registry, and send a Login request.
    - When user hits disconnect or selects from systray menu
        - Hang up internet connection if this is how the user's dialup connection preferences are set
        - Go to OFFLINE MODE with "explicit_disconnect" flag set
    - When connection to server is lost
        - If UI is open, show error message
        - Go to OFFLINE MODE

Panels - common functionality

The AppleSoup client is organized as a series of Panels, accessed by a button bar.



Navigation between panels occurs in one of two ways, either a user clicks on an item, for example a File on the File Clubs panel. In this case the appropriate panel opens to view the selected item. Alternatively the user can click on one of the buttons in the button bar. In this case the panel opens with the previously viewed contents.

Clicking on the Panel Graphic does nothing.

Feedback goes to a predefined URL.

There is a status bar at the bottom which shows Connected or Disconnected, and may give other information about network activity.

Search Interface - see FileClubs panel:Search

Menus

The window also contains a menu bar organized as follows.

Window Controls

- ■ Brings up the usual menu for all applications.
- X: Closes the UI, but minimizes as icon in the System Tray, leaving the server connection open.
- _: Does exactly the same thing
- Full Screen: Does what you would expect in Windows

FileClub Menu

- New File Club - brings up the New File Club dialog
- ***Beta-2:*** **Page Setup: Brings up the Page Setup dialog (what do we need to specify here, is this standard?)**
- ***Beta-2:*** **Print: Brings up a standard print dialog, at the moment I don't know of any AppleSoup specific stuff to go here.**
- Disconnect (or Connect): Causes the client to drop the connection to the server or to Open it.
- File Club Info - brings up File Club Info Panel, should be greyed out when nothing selected
- File Info - brings up File Info Panel, should be greyed out when nothing selected
- **Minimize and Exit need thought (Mitra)**
- Exit: Causes the client panel to close, but leaves the connection to the server open in order to serve files, AppleSoup should minimize as an icon in the System Tray.

Tools Menu

- ***Future:*** **View Full Screen: Displays the Full Screen Mode**
- **Delete File - deletes the selected file from the collection, should be greyed out when no file selected.**
- **Delete Club - deletes the collection - this can mean deleting your own collection from a hard disk, or deleting a subscribed to or clicked-on collection. In the latter two cases, the files in the collection should be deleted from the Downloads directory unless they are in another collection.**
- **From Beta: Change Password: This brings up the Change Password Dialog**
  - **This panel asks for user name, old password and new password twice - it has a button to submit and another for "I've forgotten my password".**
  - **If "I've forgotten my password" is clicked then it should work the same way as on**

the Login panel.

- **When submit is clicked, the two new passwords are compared, and if they match sent to the server in a Registration_Request.**

- **From Beta:3: Settings (or Preferences on the Mac): The options menu item will bring up a an options panel where people can set various options for the application. Options to be set include: Mitra TODO: finish this list and spec panel**
    - **automatic download size - the maximum size of a file that should be automatically downloaded when the View panel is looking at, or about to look at, it.**
    - **advanced features (including Search check-boxes)**
- **From Beta 3: Email Addresses:**
    - **This sends a command Email_Address to the server, and lists the email addresses in a dialog box or panel which has:**
        - **A line or two of text saying "This allows you to register extra email addresses for yourself. These addresses are used to filter out messages from other AppleSoup users so that you receive them inside AppleSoup rather than via email notification and a browser."**
        - **A list of existing email addresses - with:**
            - **some kind of control (check box?) to delete them, the client should not allow the last validated email address to be removed.**
            - **Email addresses which have not yet been validated should be shown differently and if there are any then an explanation given.**
        - **And a box labelled Add in which to enter any additional email address**
        - **And a button "Update" which when clicked performs minor checks for a reasonable email address, which if passed send an Email_Address? add=foo@bar.com&delete=js@bc.de&delete=kl@mn.op message to the server.**
        - **The client will then recieve back a new list of addresses and can update the display.**

Help Menu

- **AppleSoup home: Open the browser at: http://www.applesoup.com/home.htm**
- **AppleSoup Help: Open the browser at: http://www.applesoup.com/help.htm? userid=xxx&panel=yyy**
    - **send as much information as the client has in order to make it easier to give help.**
- **Feedback: Opens the browser at http://www.applesoup.com/help.htm? userid=xxx&panel=yyy**
    - **send the same information as for "Help".**
- **Privacy Policy: Opens the browser at http://www.applesoup.com/privacy.htm**
- About AppleSoup: Displays the AppleSoup version (build) number and logo

If it is not possible to connect to the AppleSoup server the client opens in a local mode without such connection. The mode (connected or local or connecting) is shown in the Status bar.

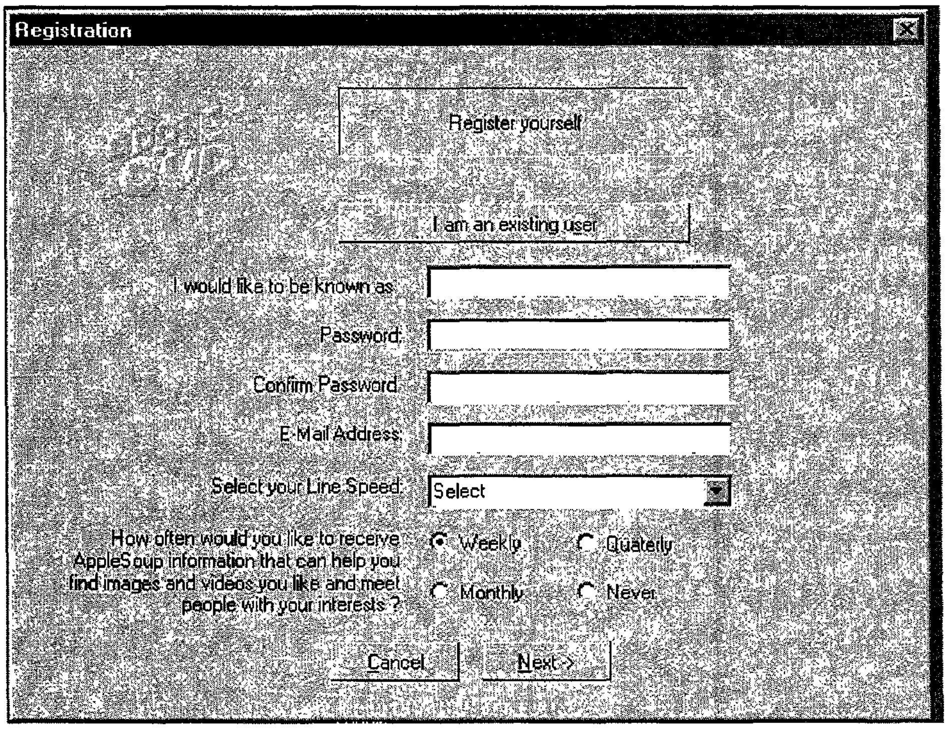
Colors

The screen shots below may not show the correct colors yet, colors should be.

- panel color R221 G221 B221

- panel highlight R255 G255 B255
- panel bevel shadow R153 G153 B153 and R0 G0 B0
- panel "inside" areas like for the menu tree and thumbnail area R191 G191 B191
- item highlight color R119 G119 B119
- window bar R170 G170 B170

Registration Panel



I would like to be known as:

Password:

Confirm Password:

E-Mail Address:

Select your Line Speed: Select

How often would you like to receive AppleSoup information that can help you find images and videos you like and meet people with your interests ? ● Weekly ○ Quaterly ○ Monthly ○ Never

This screen appears when a user starts up the AppleSoup client for the first time.The AppleSoup client checks to see if the computer already has a user configured. If not it prompts with the registration panel.

The panel contains a button - "I am an existing user", which goes to the Login panel.

The connection speed is from a list of values which are hard coded into the install script. The values are: "Not Known",14.4 modem, 28.8 modem, 33.6 modem, 56k modem, 56k ISDN, 128K ISDN, Cable, DSL, T1, T3 or more.

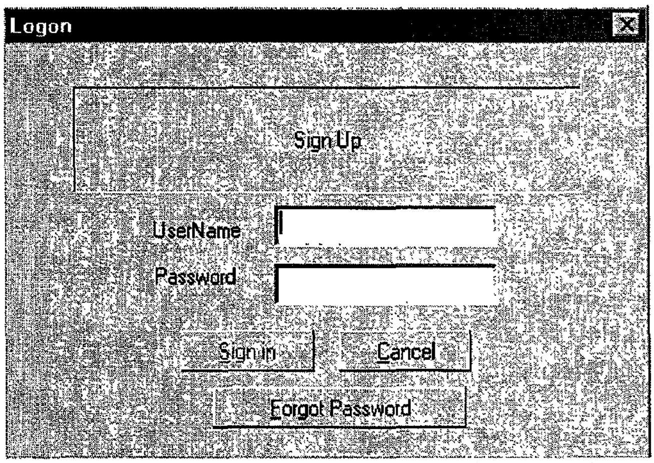
The mail-frequency radio-boxes allow the user to choose how often to get updates from AppleSoup.

When the "Register AppleSoup" button is pressed.

- Checks are made, specifically.
  - If the password and the re-enter password are not the same a MessageBox appears and they have to reenter.
  - **If the Email address looks strange (doesn't match the regexp [-a-zA-Z0-9_.]+@[a-zA-Z0-9]+.([a-zA-Z0-9]+.)+) then the user is prompted to reenter.**
  - The userid is lowercased, if it doesn't match [-@.a-zA-Z0-9]+ then it is rejected.
- Otherwise the data is sent to the server in a Registration_Request.

- If this request fails.
  - If the string reason is "NAME_FAIL" then a dialog box is presented saying "This username is already in use, please select another one"
- If this request succeeds the Login dialog is opened

Login Screen



This screen is reached when a registered user opens the AppleSoup UI (it doesn't appear if AppleSoup is started in the systray), or when a user hits "I'm already registered" from the Registration screen.

Change from screen shot:

"New User" goes to the Registration screen

The user enters their AppleSoup id and password and hits "Login". This information is passed to the server for authentication. If the server authenticates the user to be an AppleSoup user then they can enter the AppleSoup client and are shown the FileClubs panel.

As the interaction proceeds with the server it is indicated in the upper panel replacing Sign Up"

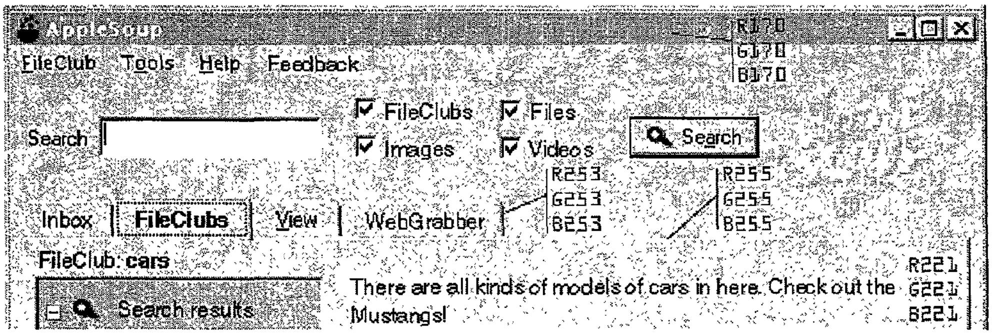
If they enter the wrong password, they will be prompted one or more times to get it right, the string for this will be supplied by the server. .

If they have forgotten their password they can hit the "Forgot Password" button, in which case the client will send "Forgot_Password" to the server, which will email the password at their email address. The client should put up a dialog box to say this is being done before exiting.

FileClubs Panel

The File Clubs Panel appears when:

- after the user logs in,

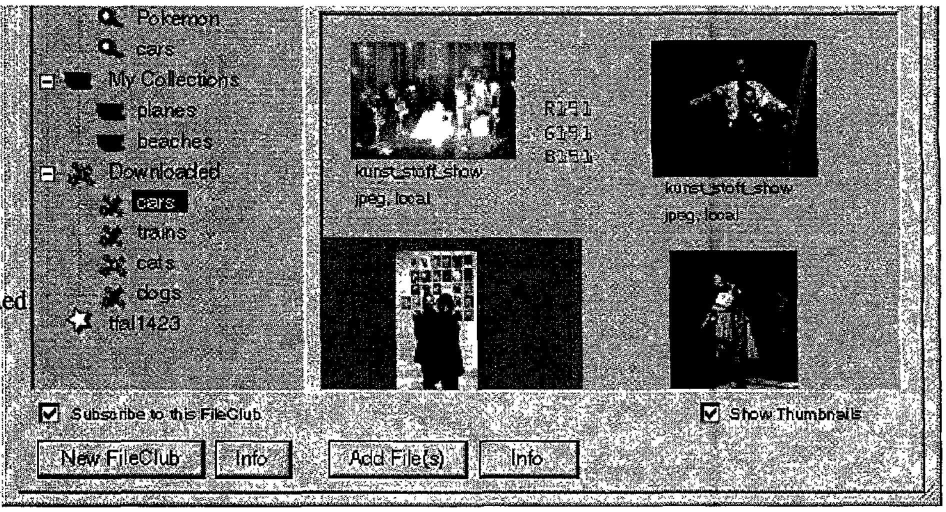


FileClub: cars

There are all kinds of models of cars in here. Check out the Mustangs!

in, this is the default panel shown, with "My File Clubs" expanded

- the File Clubs button is pushed on the Features row.



- **if a member's name is clicked on from any of several places it might be listed (e.g. search results).**
- when the Search button is clicked and a search starts

This panel is the main interface panel. It shows a list of file clubs and information about the contents of one club.

Changes from what is shown

- **Add a More button to the right of Info, with space for text "xxx results shown of yyy"**
- It should be "My File Clubs" not "My Collections"
- "Add File(s)" should be "Add Local File(s)"

Above the Bar on the Left is shown the name of the Collection being viewed, and above the Bar on the Right is shown the description of that club

From Beta 3: Below the Bar on the Right is a checkbox to switch between List and Thumbnail modes.

Left Hand Side - Explorer Bar

The Explorer Bar is 2D scrollable, and contains the following top level items.

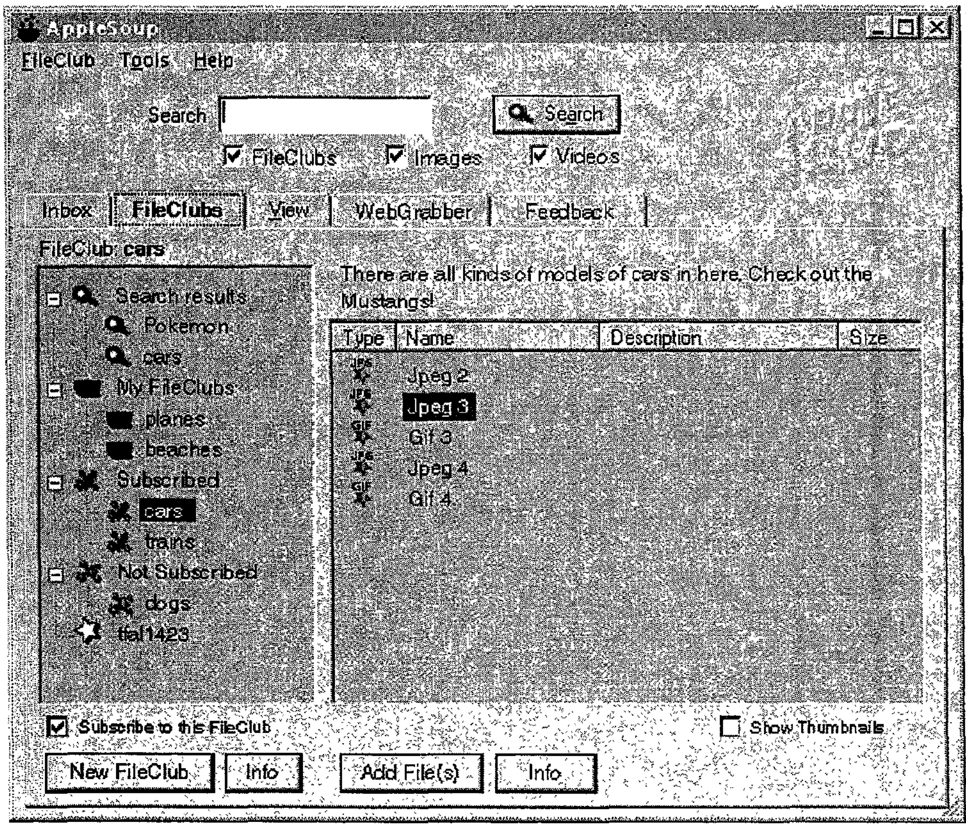
- My Clubs: This contains one Folder for each FileClub the user manages i.e. those on the user's disk.
- Subscribed: This contains one Folder for each FileClub the user is subscribed to.
- Not Subscribed: This contains one Folder for each FileClub the user is not subscribed to, this persists only for the current session. **Open question - I think this is not needed, omit for now.**
- Search Results: This, and the sub-items are shown with a magnifying glass icon. It contains one sub-item for each search listed by the search term used. Each sub-item contains one Folder

for each Collection returned in the search.

- **Note new icon:** Web Grabs: Contains one Folder for each URL that has been Web-Grabbed, listed by the URL.
- Other users. Contains one Folder for each user that has been clicked on. Each of these Folders contains a list of that user's collections.

There is the following functionality.

- When a file club folder is selected, the club is shown in the Right-Hand-Side
- *From Beta 1:* Below the explorer bar is a check box, which shows if the club is subscribed to or not, if it is changed the Client should subscribe/unsubscribe to the club.
- **Right Clicking should bring up a menu of things that can be done to a Collection - specifically.**
    - o **Info brings up the File Club Info Panel**
    - o **Delete - deletes the entire collection**
    - o **Add Local Files - adds files to a collection (disabled if it is not the user's collection)**
    - o **Fetch All - downloads all files in the collection (disabled if it is the user's collection, or they are all downloaded already)**
    - o **There may be more items here later.**
- Below this, a button to create a new File Club - this brings up the New File Club panel
- and an "Info" button that brings up the File Club Info panel.



There are all kinds of models of cars in here. Check out the Mustangs!

Type	Name	Description	Size
JPG	Jpeg 2		
JPG	Jpeg 3		
GIF	Gif 3		
JPG	Jpeg 4		
GIF	Gif 4		

Right Hand Side - File List Mode

This displays the details of files present in that file club, it shows a list of the files in the club, along with information about the file, this information should be the Name, Owner, Type, Description, Local, Size **and Date added.** Type is indicated by an icon: AVI GIF JPG MPG MOV

The headings should be **sortable** and resizable and the whole component scrolls in 2D

If possible (otherwise defer to a later version) the description should be clickable and then editable.

Right Hand Side - Thumbnail mode

In Thumbnail mode (shown above),

- it shows thumnails from the collection,
- with one line giving their description,
- and a second line giving the abbreviated type, and whether local or not, and the size **(not shown above)**
- When a file is clicked its background changes color

User Interface- either Mode

- If the user selects a file, in either mode, and clicks on it, the View panel opens up displaying the selected file
- **If a user right-clicks on a file, they should get a menu containing:**
  - **Info - brings up the File Info Panel.**
  - **Delete - deletes the file from the collection, disabled unless its the user's fileclub.**
- **There is a button which shows "Add Local File(s)" for a local collection and "Fetch All" for a Search or someone else's collection.**
  - **"Add Local File(s)", which will bring up the Add Local Files panel which allows a selection of multiple files from the user's hard disk. Note that the same rules for adding files that clash apply as defined in View Panel.**
  - **"Fetch All" , causes all the files to be downloaded.**
- **If this is a search there is a button "More" which will fetch more search results if there are any (it should be disabled if there are no more results).**
- If the user clicks on "Info" then the File Info Panel is opened for the selected file.

Search Interface

The Search Interface consists of a text box, and - if expert search is turned on in Preferences - a series of check boxes. The user can enter text into the box and hit return or click Search. In this case a Search_Request is sent to the server, and the File Clubs panel opens with a new Search item added.

The search lists the various media files that match the search criteria. (What exactly it means to match a query is something that we will research). The search will be limited to find the first N results (N is a parameter in the server environment).

If a search is repeated, it should replace the existing entry.

The search is shown as a sub-item of Search named by the search term used. Each sub-item contains one Folder for each Collection returned in the search, **these are also shown on the right-hand-side.**

The search results should be shown on the right-hand-side, even if there are no results, i.e. an empty list.

If there are no results, then the search should be tried again with spaces removed - e.g. a search for "South Park" that returns 0 results should be repeated for "southpark"

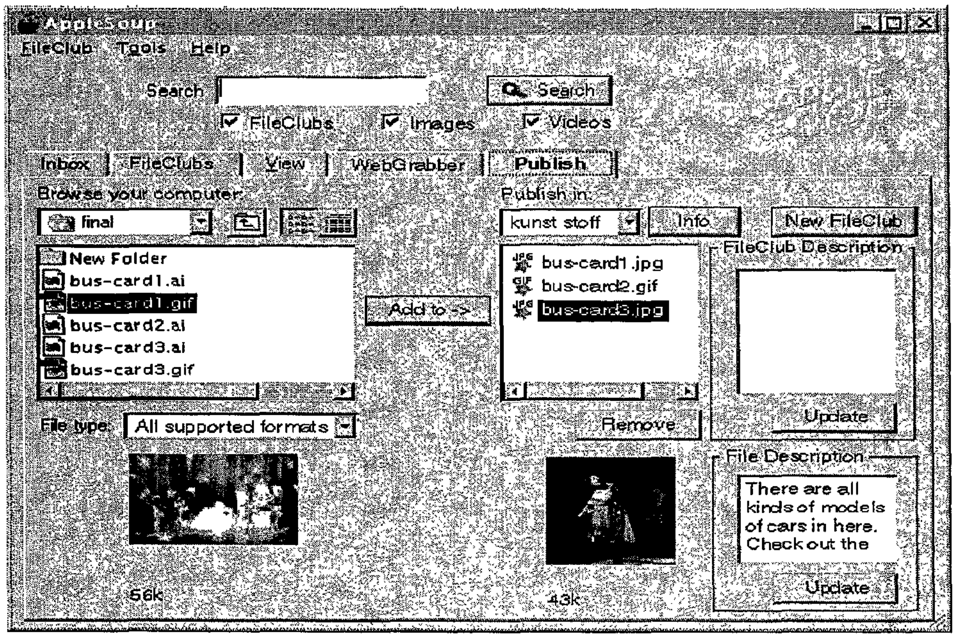
If the server returns less than the full set of search results then there should be text to the left of the More button saying "xxx results of yyy shown" and the More button should be enabled. The More button will fetch more results to be appended to the end of the list.

The client initially displays the results in the order returned by the server to allow the server to be clever about ranking results. The user can of course sort the results by any of the fields shown.

The search box is not cleared, this allows the user to edit the text and send again.

The buttons for "File Clubs", "Images" and "Videos" are initially checked and can be unchecked to restrict what gets returned.

Add Local Files Panel



final

- New Folder
- bus-card1.ai
- bus-card1.gif
- bus-card2.ai
- bus-card3.ai
- bus-card3.gif

File type: All supported formats

Publish in: kunst stoff

- bus-card1.jpg
- bus-card2.gif
- bus-card3.jpg

File Description: There are all kinds of models of cars in here. Check out the

This file is accessed via the "Add Local File(s)" button on the FileClubs panel or Right-Clicking on a collection and choosing "Add Local Files"

Changes from screen shot:

Name should be "Add Local Files"

The drop-down on the right allows selection of one of the user's file clubs.

The New Folder button on its right goes to the New File Club panel.

The panel on the right lists all the files in the File Club.

The group of file selection items on the left are standard windows items behaving in the normal way. One or more files can be selected at a time.

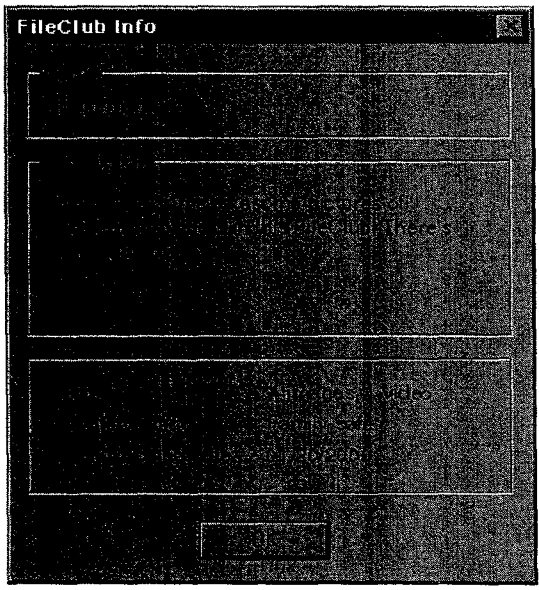
"Add to" will add the files selected to the File Club, and should be greyed out when none are selected.

Cancel will cancel an adding operation.

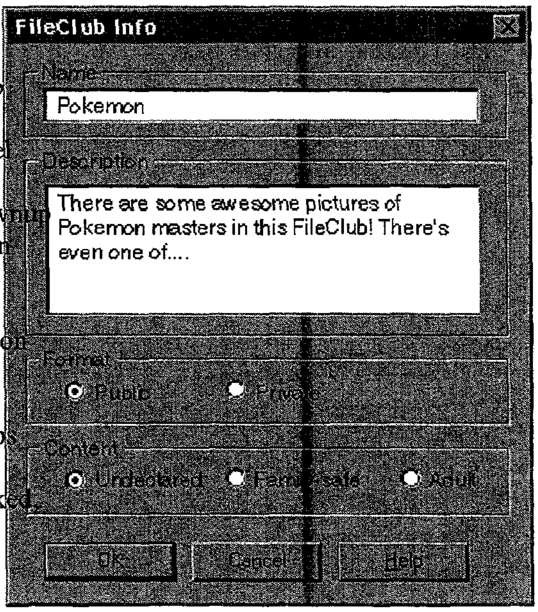
Below the selection boxes are two preview windows which will show the most recent file selected on either side.

"Remove" removes the highlighted file or files from the File Club, it should be grey when none are selected.

File Club Info Panels



The File Club Info Panel is shown up when the Info Button for File Clubs is clicked, **or via the Right-Click menus on clubs.**



Name: Pokemon

Description: There are some awesome pictures of Pokemon masters in this FileClub! There's even one of....

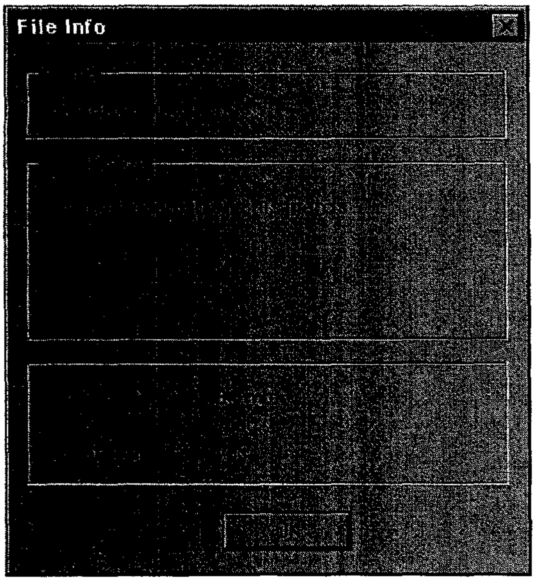
Format: ● Public ○ Private

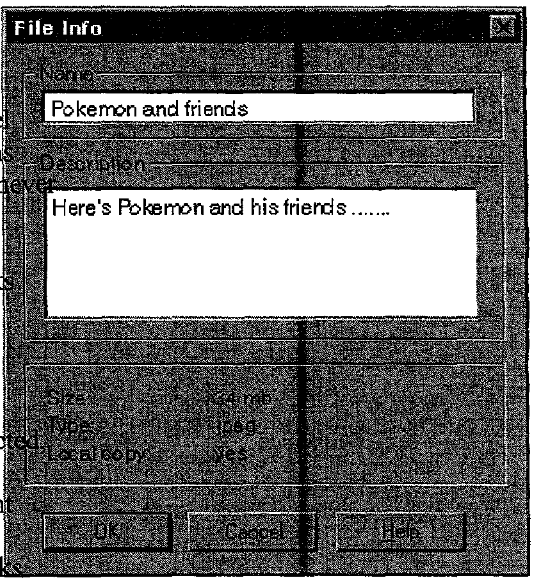
Content: ● Undeclared ○ Family-safe ○ Adult

If the File Club is owned by this user then the editable version is shown.

The FileClub panel contains the Name, the Description, Number of images and videos, Content-Rating, Last Modified Date, and **(not shown)** the Owner and whether subscribed or not and whether public or private.

File Info Panel





Pokemon and friends

Here's Pokemon and his friends ....

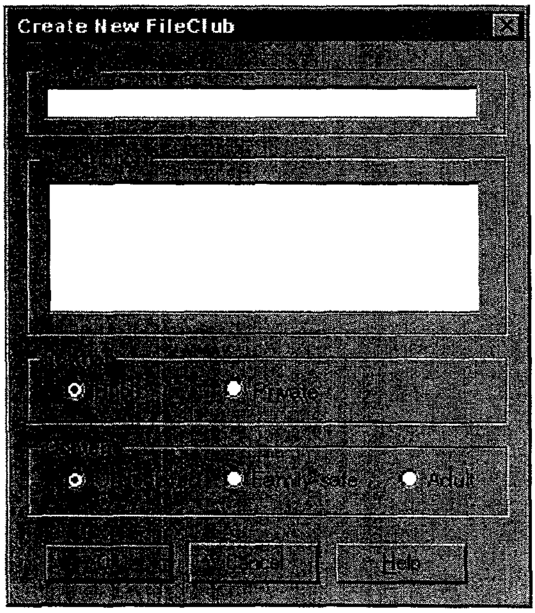
Size: [illegible]
Type: jpeg
Local copy: yes

The File Info Panel opens whenever the user clicks Info with a File selected or Right-Clicks a file and selects Info.

If the File is in a Collection belonging to this user then the editable version is shown, otherwise the non-editable version is shown.

The panels show the Name, Description, Size, Type. The non-editable panel also shows if a Local Copy exists. **This is also shown for the editable version, which is incorrect.**The panels also show if the File is adult. **(not shown on screen shots).**

New File Club Panel



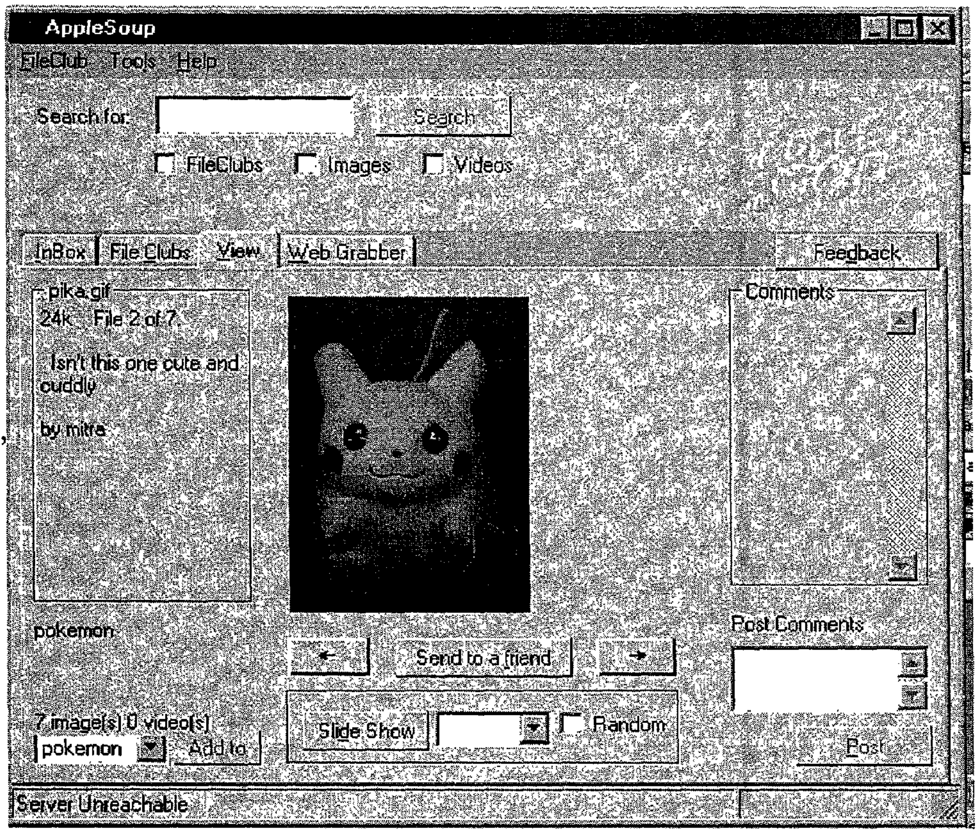
This panel is shown when the user clicks New File Club on the File Clubs panel or from the FileClubs menu or clicks the New-Folder icon in the Add Local Files Panel. It allows for entry of a Name and Description, Whether its public or Private, and a rating.

When OK is hit, this should create a directory, but nothing should be sent to the server until some files are added to the FileClub.

A confirmation dialog should be shown.

View Panel

We refer to the list of files as the media file set in the comments below. This panel logically contains two sets of information and functionality, that associated with the media files and that associated with comments that relate to the media files.



pika.gif
24k File 2 of 7

Isn't this one cute and cuddly

by mitra

pokemon

7 image(s) 0 video(s)

Server Unreachable

The view panel displays a single Media file, but with an assumption that the file is part of a list, and therefore it is possible to move backwards and forwards through the list.

The panel is reached from anywhere that a user selects one file from a list, specifically: the File List panel.

Changes from image shown.

- Place a Delete button next to the Add-To button.
- ***From Beta-1*: Add a line under description for a checkbox for Subscription**
- So the vertical order is now ... FileClub Name; FileClub description; Number of images and videos; Subscription; Drop down and Addto and delete;

Media File Presentation

The chosen file is presented using a viewer chosen by the following algorithm: **Mitra:ToDo - spec algorithm describe future**

From Version1: The client will be smart about pre-downloading the images and comments, for example it might sequentially fetch files and comments in the direction the user is clicking, or fetch comments for a range of files.

User Interface

- Video Controls
  - For at least Beta-0 we will use the Video controls that come with Windows standard API calls.
  - The control and progress bar only appears on the video
  - The progress bar should stretch as the panel is stretched, i.e. it should go from the buttons all the way to the edge on the right.
  - Moving the Progress bar around should change the position in the video (save for a later version if this is tough).
  - Play button turns into Pause "||" when it is playing, pressing pause stops the play but leaves the position at the same place and the button turns back to Play.
  - The Stop button causes it to stop and reset to the beginning.
- The buttons have different images for depressed, which **Beverly will send,**
- If possible, the description should be editable if the files are your own.
- The **"->"** and **"<-"** buttons cause the next, and previous media files of the media file set to be displayed. These buttons loop around between the first and last images. Space, tab, right and down arrow should shortcut to next while shift-tab, up and left-arrow should shortcut to previous.
- If "Random" is checked then the slideshow should pick files in a random order.
- The "Slideshow" button cycles through the media file set displays the next media file after N seconds.
  - N is initially a constant, *(from Beta-3 it is set in preferences)*
  - It should display the first one immediately.
  - When depressed, it should change text to Stop.
  - **Next "<-" and Previous "->" should be changed to Faster and Slower, and cause the speed of presentation to speed up or slow down.**
  - Random should be disabled.

- There is a drop-down menu of the file clubs. One of these can be selected. When one has been selected the "Add to A File Club" button is activated, and when clicked, causes the media file displayed to be added to a file club. This occurs on the client machine, and then appropriate Upload_Collection messages are sent to the server.
  - **File Clashes:** When a user tries to do this, there are four possible cases:
    1. There is a file there with the same hash and the same name: In this case do nothing, but tell the user. Status="File xxx was already in File Club yyy"
    2. There is a file there with the same hash, but a different name: Do nothing, tell the user it is there already, and what name it has. Status="File xxx was already in Club yyy with the name zzzz"
    3. There is a file with the same name, but different hash: Give the user the choice of overwriting the old file, or of selecting a new name to store this image as.
    4. There is no clash: Just store the file. Display status="File xxx added to File Club yyy"
- **The Delete Button when pressed should delete the file from this computer, and if it is**

currently showing your own collection should remove from the collection.

- *Version 1: The "Collections" button goes to the File Clubs panel with a list of collections this file is a member of.*
- *Beta-3. The "Next:" label occurs over a thumbnail of the next media file to be displayed.*
- When "Send to a Friend" button is pressed the "Send to a Friend" dialogue appears:
- There is an indication of which file (X) out of the total number (Y) is being presented.
- Potential future plans .... The "View Full Screen" button **(Not shown in screen shot)** opens the Full Screen mode with this image.

Comments Presentation and Maintenance (right side)

This portion of the screen provides comments on the media file currently being shown.

As a media file is displayed, the comments, that are associated with that media file, are displayed on the "Comments" panel in the order returned by the server, which is roughly reverse-chronological order i.e. most recently entered first, but may vary if comments for this file are retrieved from other collections.

Until Beta-3, once comments are read for a file, they will not be reread until for some other reason the client requests file information on the file.

From Beta 3: The client and server will have to be smart about when to update comments. **Mitra ToDo - spec this**

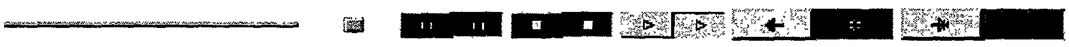
Each comment has: user id, date of posting and the text of the comment. If there are more comments than there is space for, then scrolling will display them.

The commenter ids are active. If the member clicks on the commenter id, the File_Clubs window is opened showing the Collections of the commenter.

There is a "Post Comments" text entry window for new comments. When the "Post" button is pressed the comments in the "Post Comments" text field (if not null or blank) are posted to that media file, using the Post_Comment message, and immediately displayed at the top of the list of comments.

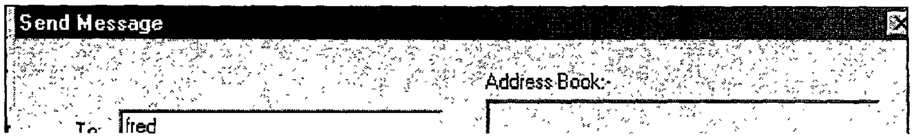
User Interface Icons etc

Some of the icons etc used here, with the depressed versions as well, are ....



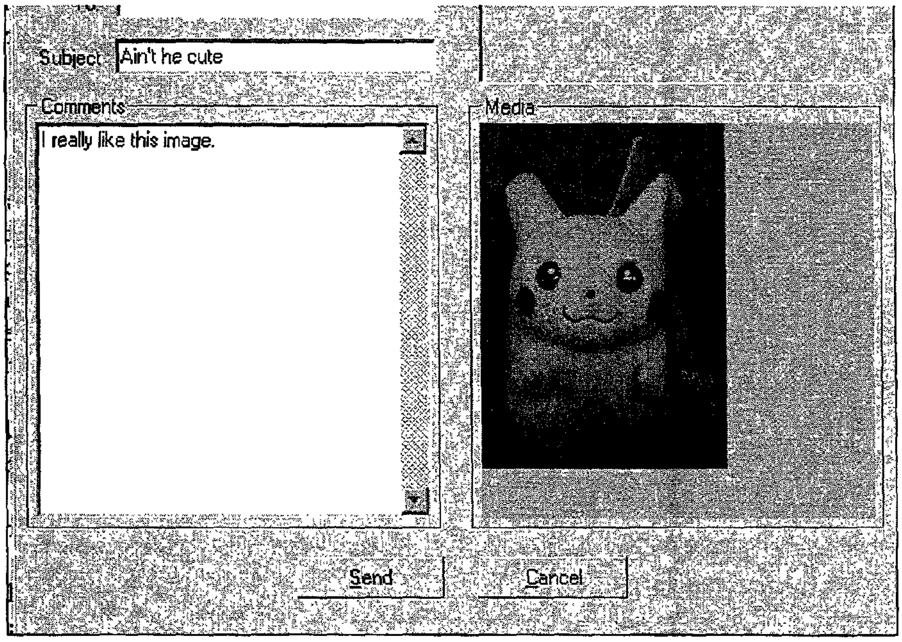
Send to a Friend

This is reached when "Send to a Friend" is clicked in the View panel.


Send Message

To: fred

in the View panel.



Subject: Ain't he cute

Comments: I really like this image.

Add an arrow that when clicked adds names from address book to comma seperated list in To: field.

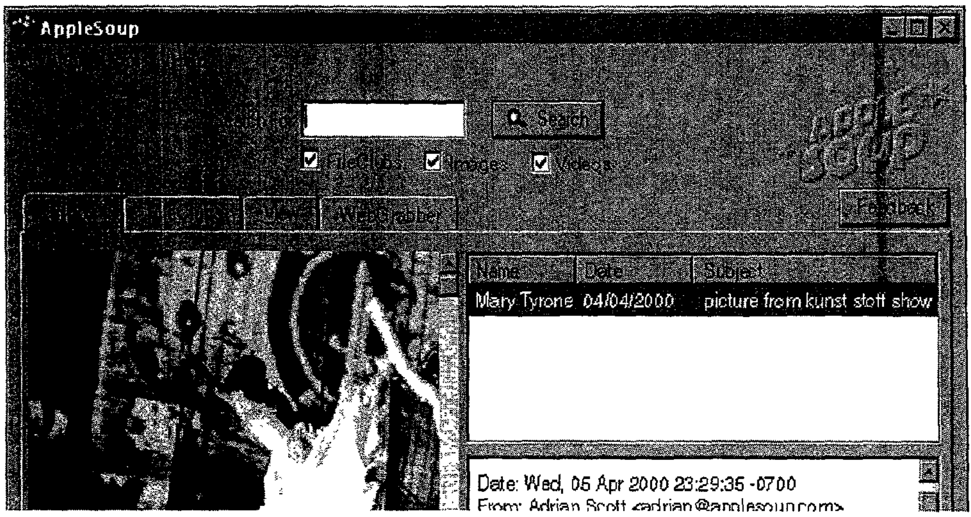
The same media file that is presented on the View Panel, appears in the presentation area of this panel. The member can type in a new e-mail address in the "To:" field and/or can select any other friends of the list (ListBox) to the right. The member enters a subject which should not be null and can add comments (optional).

When the Send button is pressed, the Send_Message command is sent.

If it fails, then the panel is represented with the error addresses highlighted and a message telling the user to change them.

If it succeeds, then any newly entered email address is added to the address book. The panel then disappears and the View Panel reappears with the same media file displayed.

Inbox Panel



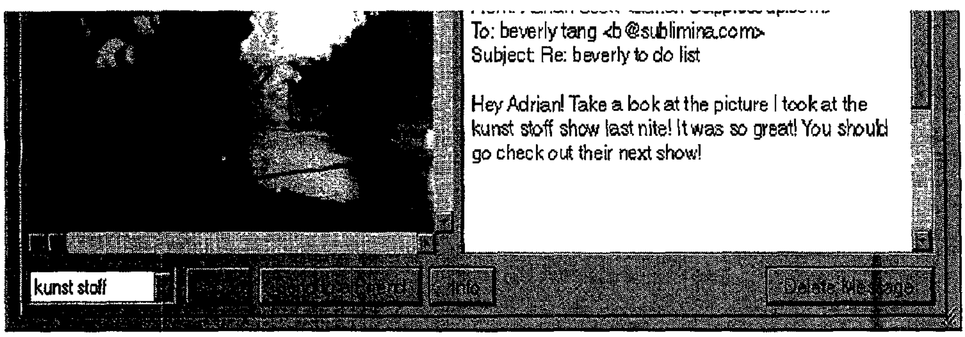
Name	Date	Subject
Mary Tyrone	04/04/2000	picture from kunst stoft show

Date: Wed, 05 Apr 2000 23:29:35 -0700
From: Adrian Scott <adrian@applesoup.com>

The inbox panel shows a list of messages for the user.

differences from what is shown

Image should be on right, list and text on



To: beverly tang <b @sublimina.com>
Subject Re: beverly to do list

Hey Adrian! Take a book at the picture I took at the kunst stoff show last nite! It was so great! You should go check out their next show!

text on left.

For each message it lists: Who it is From, **Date sent,** Subject, messages that are unread are shown in Bold.

The list box is 2D scrollable with resizable and sortable columns.

The vertical and horizontal seperators between components should be adjustable.

Clicking on a selected message in the Inbox Panel opens it in the lower panel. **"From", "Date", "To" and "Subject" at the top**, and then thetext accompanying the message.

There should be a button for Delete, and button and **drop-down for adding to a fileclub**, and a button for Send-To-A-Friend. These behave the same as for the View Panel.

Help Us Panel

The feedback panel is made active by pressing the "Help Us" button. This goes to a browser window at a pre-defined URL. As much useful information as possible should be embedded in this URL, for example, userid, which panel active, which collection, which file. *prior to Beta-4 this will be a common browser window, after it will be an HTML control embedded in the client.*

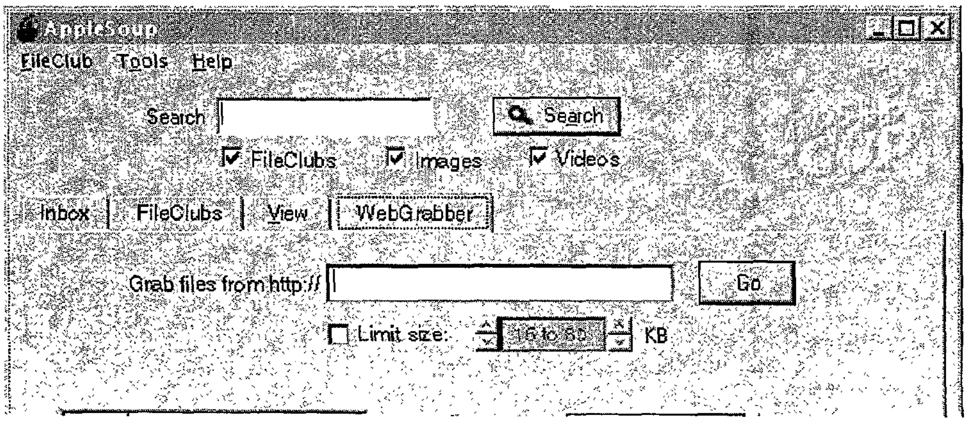
This URL will contain an HTML screen that requests feedback.

Web Grabber Panel

The Web Grabber grabs images off a Web site.

Changes from screen shot

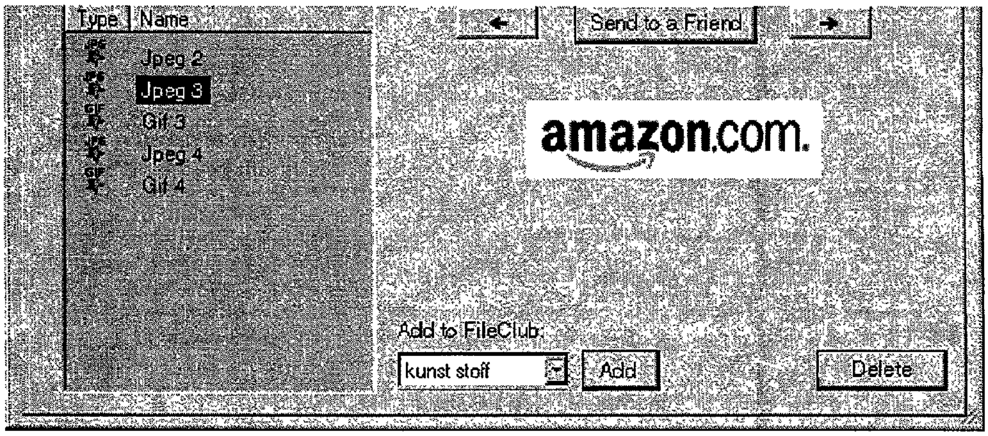
- this change to view files



AppleSoup
FileClub Tools Help
Search
Search
☑ FileClubs ☑ Images ☑ Videos
Inbox | FileClubs | View | WebGrabber
Grab files from http://
Go
□ Limit size: 16 to 50 KB

view files can wait till Beta-1

- use two spinners with text "to" in middle.



The files should be placed in a File Club with a name being that of the URL, and initially set to Private. There should be a progress bar to show how it is doing. Clubs created like this are NOT shared.

From Beta-2: Since the URL is likely to be from an already open browser window, the client should, if possible, look in the Browser cache for images first.

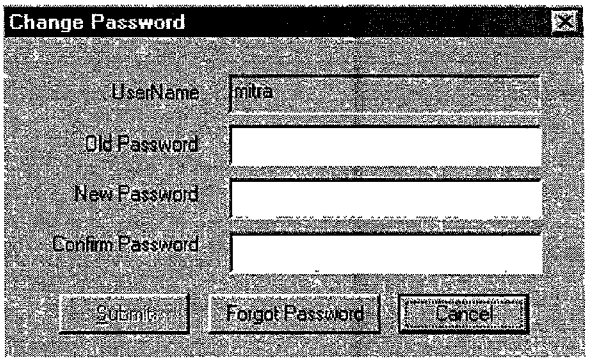
Change Password Dialog

This dialog appears when the user selects "Change Password" from the tools menu.

The user has to enter their Old password, and the new one twice.

Forgot Password works as it does on the Login screen.

When Submit is hit then.


Change Password

UserName mitra

If the New and Confirmed passwords don't match then the user should be prompted to re-enter them.

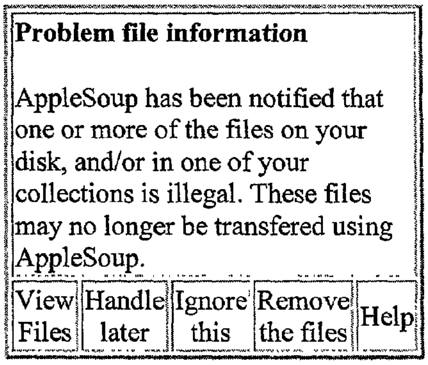
If the Old Password doesn't match that saved then the user should be prompted to re-enter it.

Otherwise the message is sent to the server (as an authenticated registration request). **The status line should be updated and the dialog closed.**

Copyright Dialog

Do not implement this yet

This dialog is generated whenever a <COPYRIGHT> message is received from the server. First the client stores the copyright status on all the files. The client displays dialog saying



Problem file information

AppleSoup has been notified that one or more of the files on your disk, and/or in one of your collections is illegal. These files may no longer be transfered using AppleSoup.

The dialog should say illegal for reason=3 and copyright for reason=2

- *View Files should open them in the Viewer with < and > cycling through the files*
- *Handle Later should just ignore this message - the user will be notified next time they log in.*
- *Ignore this should ignore the message and record that the user has already been notified so that the message can be ignored in future.*
- *Remove the files should delete from the disk and any collections.*
- *Help should go to a predefined URL http://www.applesoup.com/help/copyright_notification.jsp?hash=1a2b3c&reason=2*

Full Screen Mode

Future Plans - not to be implemented yet.

Full Screen mode is accessed via button on the View Panel. The image currently being viewed is shown full-screen with all the user-interface hidden. Clicking, or Right-Clicking on the screen will bring up control buttons **(Bev to supply design)** as will using any key not specified as a shortcut. **Initially base these buttons on those in the View Panel, but replace "View Full Screen" with "Exit Full Screen".**

In Full Screen Mode it should not cycle to the next image until the image has been downloaded.

Windows Integration (From Beta1)

Its is AppleSoup's goal to encourage people to use AppleSoup by making it easy to work with their existing applications, for example Windows Explorer, IE, HTML editors, etc etc. To achieve this, tight integration with Windows is needed. As many of the following as possible should be implemented.

- Button on IE, that when pushed opens the Web Grabber panel with the URL filled in from that of the page being viewed in IE.
- Right Click on an image in IE, the user should have the option to Add the image to a AppleSoup file club.
- Right Click on an image in AppleSoup (View or FileClubs or Inbox or WebGrabber), the user should:
    - o if they have the appropriate editors registered for that mime type - be able to Edit it in

their image editor
    - be able to do a "Send To" as in Windows Explorer
    - be able to Copy
- Drag an image or a HTML file or a shortcut from any of:
    - IE, Windows Explorer, or any other standard windows situation
    - onto AppleSoup in the System Tray or directly into a File Club
- Dragging an image out of AppleSoup into a Windows application - for example into Explorer, or an editor, or MS Word.
- Ctrl-C or Right-click followed by Copy on a file in either the View panel or any of the listing panels (File List, or Search)
- Ctrl-V or Right-click Paste into a File Club or View panel.
- Note that however a file is added, the rules specified under View Panel for File Clashes apply.

Some of these may not be possible, and there may be other obvious integrations with Windows.

On a Mac or Linux similar, but different conventions should be followed.

System Tray

When operating from the System Tray, then a minimal UI is available.

Clicking or Double-Clicking on the Icon should open the AppleSoup Application.

When a message arrives for the user, then the icon should change to:

Right-Clicking on the Icon brings up a menu.

- Disconnect (or Connect): Causes the connection to be dropped, or opened
- Open AppleSoup: Opens the UI
- You have xx Messages: Shows how many messages are waiting to be viewed.

File and Data storage Structures

The AppleSoup client keeps information on the client in a number of locations.

- C:/Program Files/AppleSoup (default overridable at install time), contains all executables, and other files installed by the installer that have nothing to do with the data. Files in here should not be changed after the install, except when an upgrade is performed.
- Windows Registry containing a set of parameters for the client.
    - [HKEY_CURRENT_USER\Software\AppleSoup]
      ```
      "AppleSoup"="C:\\Program Files\\AppleSoup"
      "ClublistPath"="C:\\AppleSoup\\Clubs"
      "Download"="C:\\Applesoup\\Downloads"
      "Server"="128.12.1.44"
      "Port"="80"
      "ListenerPort"="3300"
      "Language"="EnglishUS"
      ```
- C:/AppleSoup (default overridable at install time) contains all data related to the application.
    - index.xml - file containing information on the collections, description etc.

        - This file should also contain the list of friends.
        - In the live version, it can be a hidden file.
        - The client team have freedom to specify this file as fit, based on the XML structures transferred from the server. After Beta the example file should be attached here.
    - "Downloads" directory contains all the files retrieved from the server
        - tajmahal.gif - typical file
        - tajmahal.02.gif - another file with the same name
        - tajmahal.thumb.gif - a thumbnail of tajmahal.gif (*Beta3)*
        - tajmahal.02.thumb.gif - a thumbnail of a secondary file (*Beta3)*
    - "Clubs" directory containing one subdirectory named for each of the local FileClubs
        - "India" - a typical FileClub
            - elephants.jpg - a typical file

ActiveX Controls

Control Selection Criteria

We will want to use some third-party controls. Let's use the following criteria to guide the selection of controls:

- Standardness and Reliability -- e.g. use Microsoft standard controls where possible
- Low file size -- where possible we want to keep the file sizes low
- Simplicity -- we don't need controls that do everything. For example, in some cases there are Microsoft controls that do everything but there are other Microsoft or non-Microsoft controls that do most of what we need with a much smaller file size and reduced, but adequate, feature set.

Controls Needed

We should research and select third-party controls for the following capabilites, where possible:

- Displaying images -- jpeg, gif and animated gif. The AnimGif OCX used in the prototype may meet our needs here.
- Viewing videos -- avi, mpeg and if easy MOV. The MCI API calls may meet our needs here.
- MD5 hash value generation -- these has values are used to uniquely identify the media files.
- HTTP & SSL communications -- presumably we can use Microsoft controls for this. We should research this -- particularly whether it'll handle SSL encryption and whether it'll handle persistent HTTP1.1 connections and HTTP1.0 Keep-Alive connections.

Communications Protocol

The AppleSoup client will communicate with the AppleSoup server and other AppleSoup clients using a TCP/IP communications protocol. This protocol will wrap each information transfer in a suitable transmission envelope. The design of this envelope is defined in the Communications Protocol Document and the current set of Windows bindings to it is specified in common.h.

Generally an outgoing request message will contain certain data elements which are wrapped in the request envelope. The data elements will consist of a request type and other data. In response the server will reply with a message that contains a response type and associated data items. (Consequently a fuller understanding of the client can be obtained by reading the Server specification Document in conjunction with this document).

One implication of the design is that the client should have a listening process that waits for messages coming from another client.

Some specific aspects of functions are described below. Others are described above along with the panel they refer to.

Get_File

The client has to both generate and respond to Get_File commands, it should be listening on the specified port for incoming Get_File commands. The Requesting Client generates a simple GET to the first Supplying Client listed.

```
GET /get_file?hash=1234567781212332321432
Host: client.applesoup.com
Accept-Lanuage: en, fr;q=0.5
```

Note that the "Host" field here is faked!

The supplying client checks to see if it has this file, in both the Downloads directory and any Collections. If it has the file then it returns it as a standard HTTP response containing the file.

If the file is not available, or if the file is marked copyright > 0 (whether deleted or not) or there is some other problem then a HTTP error code 404 should be returned to the requesting client.

On success, the Requesting Client stores the resulting file in the downloads directory, and sends a Log_Transfer command.

On failure the Requesting Client sends a Transfer_Failed command to the server, and tries the next location offered.

From Version 1: If there are no more possible locations then the Client should notify the User (presuming this is not some automatic transfer) and then store the fact that this file was requested. At some future time, the Client can re-request from the server a list of locations to try for this file.

From Version 1: After receiving a list of locations, the selection process could involve in the future sending an ICMP (ping) packet to each of the top "n" potential locations, and using the return time (as well as the connection speed) as part of the decision of which to try a Get_File with.

Alternative methods of reaching clients behind firewalls will be explored later, for now we will assume that a client behind a proxy can only receive but not serve files.

Installation Procedures

The installation works in several steps.

First on a new client, a user downloads an installer. This installer is a self-extracting Zip file which contains:

- a small (~20k) .exe containing just the main( ) for the installer
- a .dll containing the classes that support download and installation.
    - This will be statically linked with MFC for a size of approx 280KB
    - These classes will also be used by AppleSoup for updates
    - This does not include a fancy background for the installer, which would have added another 200+KB
- a .ini containing information on what versions of what modules are needed.

The installer will launch and ask:

- where it should be installed defaulting to: C:\Program Files\AppleSoup
- where it should place shared media files, defaulting to C:\AppleSoup\Clubs
- where it should store downloaded media files, defaulting to C:\AppleSoup\Downloads

These values are stored in the Registry for future use.

The installer will check for each module whether its "Must" be present, and what the version of any current copy is. It then presents a dialogue box to the user showing each module group and checkboxes for each module. The checkboxes are filled in as follows.

	Must=0 **The module is optional, for example language fonts or a viewer**	**Must=1** **The module is required for successful functioning of AppleSoup**
No version on the user's machine	unchecked, and may be changed by a user who wants to add this functionality.	checked and cannot be changed.
Version older than Required version	checked, if the user unchecks the box, then they are warned that this may effect functionality.	
Version Newer than Required	checked but can be unchecked by the user with no warning.	
Current Version	unchecked, but can be checked by the user if they wish to reinstall for some reason.	

There are a few exceptions to the above simple rules.

The Anigif control for viewing images.

- The installable checks whether the Web Browser control is installed on the users machine. **How?**
- If installed, we do not download the cab file for "Anigif control"(control for viewing images) and create a registry entry in HKCU\Software\AppleSoup with a value "UseWebBrowser = 1".

- If not installed, we download the "Anigif control" and set the "UseWebBrowser=0".
- Depending upon the value set for "UseWebBrowser" the application will decide whether to use the Web Browser or the Anigif control.
- *Note that there is a currently unsolved problem with the Web Browser control that the image cannot be resized, until solved the Anigif control will always be downloaded.*

The file Resources.dll contains the text etc for buttons, and is language dependent, if it needs to be downloaded, then the Installation procedure will allow the user to select precisely one of the available language options.

The installer will check whether the needed MFC .dll's are present on the user's machine, if they are then smaller versions of some of the files will be downloaded, otherwise files with staticly bound MFC calls will be downloaded. **TODO:** Research is happening to determine how frequent it is that these files are available, for example on Win98 it is possible that they are always available.

The installation module will then fetch the .cab files for the modules checked, decompress and install them and delete the temporary files. It will then start AppleSoup.

TODO: The Software Update functionality will be changed after a bit more research to allow the server to specify when new versions are available and provide the ".ini" information to the client, so that the AppleSoup client can then update itself.

Future versions of the AppleSoup client will allow the client to spot the Unicode range of comments and prompt users when they need to install language fonts.

Future versions of the AppleSoup client will spot unknown media types, query the server for available viewers, prompt the user and then download new viewers.

1. A method for accessing information in a peer-to-peer network, the peerto-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, the method comprising:

selecting a first information file;

generating, using fingerprinting algorithm, a first fingerprint ID relating to the content of the first information file; and

identifying the first information file using the first fingerprint ID.

2. The method of claim 1 wherein the fingerprinting algorithm corresponds to an MD5 Message-Digest algorithm.

3. The method of claim 1 wherein the fingerprinting algorithm corresponds to a Secure Hash Algorithm (SHA1).

4. The method of claim 1 wherein the first information file is stored at a first peer device, and wherein the first information file has an associated first filename, the method comprising:

storing the first filename and first fingerprint ID at the first peer device.

5. The method of claim 4 further comprising: transmitting the first filename and the first fingerprint ID to the database system for storage therein.

6. The method of claim 5 wherein the database system corresponds to a remote database system.

7. The method of claim 1 further comprising:

selecting a second information file having content identical to the first information file;

applying the fingerprinting algorithm to the content of the second information file to thereby generate an identical first fingerprint ID to that of the first information file; and

identifying both the first and the second information file using the first unique fingerprint ID.

8. The method of claim 7 wherein the first information file is stored at a first peer device, and has a first associated filename, and wherein the second information file is stored at a second peer device, and has a has second associated filename, the method further comprising:

storing the first associated filename and first fingerprint ID associated with the first information file in the database system; and

storing the second associated filename and first fingerprint ID associated with the second information file in the database system.

9. A method for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective fingerprint ID associated therewith relating to its file content, the method comprising:

transmitting a first message to the database system, the first message including a search request for locating files in the network which match a first search string; and

receiving a first response from the database system, the first response including first information relating to identified files stored in the network which match the first search string;

the first information further including an associated fingerprint ID for each identified file.

10. The method of claim 9 further comprising:

transmitting a second message to the database system, the second message including a first fingerprint ID selected from the first information; and

receiving a second response from the database system in response to the second message;

the second response including second information, the second information including at least one network address corresponding to at least one peer device that has been identified as having access to at least one file corresponding to the first fingerprint ID.

11. The method of claim 10 further comprising:

transmitting a third message to a first peer device of the at least one peer devices, the third message corresponding to a request to retrieve a first file identified by the first fingerprint ID.

12. The method of claim 11 wherein the third message includes the first fingerprint ID.

13. The method of claim 11 further comprising:

receiving at least a portion of the file content of the first file from the first peer device in response to the third message.

14. A method for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective fingerprint ID associated therewith relating to its file content, the method comprising:

transmitting a first message to a first peer device, the first message corresponding to a request to retrieve a first file identified by a first fingerprint ID, wherein the first message includes the first fingerprint ID, and wherein the first fingerprint ID is different than a filename associated with the first file; and

receiving a first portion of the file content of the first file from the first peer device in response to the first message.

15. The method of claim 14 further comprising:

detecting a failure in a file transfer process associated with the first peer device;

identifying a second portion of the first file content which has not been received; and

transmitting a second message to a second peer device, the second message corresponding to a request to retrieve the second portion of the first file content identified by the first fingerprint ID, wherein the second message includes the first fingerprint ID.

16. A method for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the method comprising:

receiving file information from selected peer devices, the file information relating to shared files stored at each of the selected peer devices;

the file information including a filename for each shared file, and including a HASH ID for each shared file;

storing the file information in at least one data structure at the database system; and

identifying a desired shared file in the network using its associated HASH ID.

17. The method of claim 16 further comprising identifying an identity of a peer device using a selected HASH ID;

wherein the identified peer device has been identified as storing a file having an associated HASH ID which matches the selected HASH ID.

18. The method of claim 16 further comprising identifying a network address of a first peer device using a selected HASH ID;

wherein the first peer device has been identified as storing a file having an associated HASH ID which matches the selected HASH ID.

19. A method for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the method comprising:

receiving a first message from a first peer device, the first message including a search request for locating files in the network which match a first search string;

generating a first response to the first message, the response including a first list of file records relating to files stored in the network which match the first search string, wherein each file record includes an associated HASH ID and an associated filename; and

providing the first list of file records to the first peer device.

20. The method of claim 19 further comprising:

excluding from the first list of file records duplicate records in which multiple file records have the same associated HASH ID and filename.

21. The method of claim 19 further comprising:

receiving a second message from the first peer device in response to the first response, the second message including at least one HASH ID;

identifying, using said at least one HASH ID, at least one network address corresponding to at least one peer device which has been identified as storing at least one file corresponding to the at least one HASH ID; and

providing, to the first peer device, a second response, the second response including address information which includes at least a portion of the at least one identified network addresses.

22. A method for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the method comprising:

identifying a first network addresses corresponding to a first peer device which has been identified as storing a first information file associated with a first HASH ID;

identifying a second network addresses corresponding to a second peer device which has been identified as storing a second information file associated with the first HASH ID;

transmitting a first message to the first peer device requesting a first portion of file content of the first information file from the first peer device; and

transmitting a second message to the second peer device requesting a second portion of file content of the second information file from the second peer device.

23. The method of claim 22 wherein the first and second messages each include the first HASH ID.

24. The method of claim 22 wherein the first and second messages are initiated at substantially a same time

25. The method of claim 22 wherein the requesting of the first portion of file content from the first peer device occurs concurrently with the requesting of the second portion of file content from the second peer device.

26. The method of claim 22 further comprising:

receiving the first portion of file content from the first peer device;

receiving the second portion of file content from the second peer device;

generating a third information file which includes the first and second portion of file content, wherein the file content of the third information file is identical to the file content of the first information file.

27. The method of claim 22 further comprising:

detecting a failure in a file transfer process associated with the first peer device;

identifying a third network addresses corresponding to a third peer device which has been identified as storing a third information file associated with the first HASH ID;

transmitting a third message to the third peer device, the third message corresponding to a request to retrieve the first portion of file content from the third information file.

28. The method of claim 22 wherein the first portion of file content corresponds to a first chunk of bytes **1** to N of the first information file; and

wherein the second portion of file content corresponds to a second chunk of bytes N+1 to 2N of the second information file.

29. A method for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the method comprising:

requesting from a first plurality of peer devices a respective portion of file content from a respective information file, each respective information file being identified as having identical file content and having an identical first HASH ID being associated therewith.

30. The method of claim 29 further comprising:

receiving from at least a portion of the first plurality of peer devices respective portions of file content from the respective information file; and

reconstructing the respective portions of file content to assemble a requested information file having file content identical to that corresponding to the first HASH ID being associated therewith.

31. The method of claim 29 further comprising:

before requesting a respective portion, creating a content map of the file content associated with the first HASH ID, said content map parceling the file content into respective portions from 1 to M.

32. The method of claim 31 further comprising:

assigning at least one respective portion, from 1 to M, to a first peer device of the first plurality of peer devices to request retrieval thereof.

33. The method of claim 32 further comprising:

receiving from the first peer device the one respective portion, from 1 to M, of file content from the respective information file; and

upon retrieval of the entire one respective portion from the first peer device, updating the content map corresponding to the retrieval thereof.

34. The method of claim 33 further comprising:

upon retrieval of all respective portions, from 1 to M, of file content, reconstructing the respective portions to assemble a requested information file having file content identical to that corresponding to the first HASH ID being associated therewith.

35. The method of claim 29 further comprising:

identifying the network addresses corresponding a first plurality of peer devices, from 1 to X, each of the first plurality of peer devices being identified as storing a respective information file, each having identical file content and having an identical first HASH ID being associated therewith;

36. The method of claim 35 further comprising:

before requesting a respective portion, creating a content map of the file content associated with the first HASH ID, said content map parceling the file content into respective portions from 1 to M, where M>X.

37. A system for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, the system comprising:

at least one CPU

memory

at least one interface for communicating with other devices in the peer-to-peer network;

the system being configured or designed to select a first information file;

the system being further configured or designed to applying a fingerprinting algorithm to the content of the selected file to thereby generate a first fingerprint ID relating to the content of the first information file; and

the system being further configured or designed to identify the first information file using the first fingerprint ID.

38. The system of claim 37 wherein the fingerprinting algorithm corresponds to an MD5 Message-Digest algorithm.

39. The system of claim 37 wherein the fingerprinting algorithm corresponds to a Secure Hash Algorithm (SHA1).

40. The system of claim 37 wherein the first information file is stored at a first peer device, and wherein the first information file has an associated first filename; and

wherein the system is further configured or designed to store the first filename and first fingerprint ID at the first peer device.

41. The system of claim 40 being further configured or designed to transmit the first filename and the first fingerprint ID to the database system for storage therein.

42. The system of claim 41 wherein the database system corresponds to a remote database system.

43. The system of claim 37 being further configured or designed to select a second information file having content identical to the first information file;

the system being further configured or designed to apply the fingerprinting algorithm to the content of the second information file to thereby generate an identical first fingerprint ID to that of the first information file; and the system being further configured or designed to identify both the first and the second information file using the first unique fingerprint ID.

44. The system of claim 43 wherein the first information file is stored at a first peer device, and has a first associated filename, and wherein the second information file is stored at a second peer device, and has a has second associated filename;

the system being further configured or designed to store the first associated filename and first fingerprint ID associated with the first information file in the database system; and

the system being further configured or designed to store the second associated filename and first fingerprint ID associated with the second information file in the database system.

45. A system for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective fingerprint ID associated therewith relating to its file content, the system comprising:

at least one CPU

memory

at least one interface for communicating with other devices in the peer-to-peer network;

the system being configured or designed to transmit a first message to the database system, the first message including a search request for locating files in the network which match a first search string; and

the system being further configured or designed to receive a first response from the database system, the first response including first information relating to identified files stored in the network which match the first search string;

the first information further including an associated fingerprint ID for each identified file.

46. The system of claim 45 being further configured or designed to transmit a second message to the database system, the second message including a first fingerprint ID selected from the first information; and

the system being further configured or designed to receive a second response from the database system in response to the second message;

the second response including second information, the second information including at least one network address corresponding to at least one peer device that has been identified as having access to at least one file corresponding to the first fingerprint ID.

47. The system of claim 46 being further configured or designed to transmit a third message to a first peer device of the at least one peer devices, the third message corresponding to a request to retrieve a first file identified by the first fingerprint ID.

48. The system of claim 47 wherein the third message includes the first fingerprint ID.

49. The system of claim 47 being further configured or designed to tsb receive at least a portion of the file content of the first file from the first peer device in response to the third message.

50. A system for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective fingerprint ID associated therewith relating to its file content, the system comprising:

at least one CPU

memory

at least one interface for communicating with other devices in the peer-to-peer network;

the system being configured or designed to transmit a first message to a first peer device, the first message corresponding to a request to retrieve a first file identified by a first fingerprint ID, wherein the first message includes the first fingerprint ID, and wherein the first fingerprint ID is different than a filename associated with the first file; and

the system being further configured or designed to receive a first portion of the file content of the first file from the first peer device in response to the first message.

51. The system of claim 50 being further configured or designed to detect a failure in a file transfer process associated with the first peer device;

the system being further configured or designed to identify a second portion of the first file content which has not been received; and

the system being further configured or designed to transmit a second message to a second peer device, the second message corresponding to a request to retrieve the second portion of the first file content identified by the first fingerprint ID, wherein the second message includes the first fingerprint ID.

52. A system for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the system comprising:

at least one CPU

memory

at least one interface for communicating with other devices in the peer-to-peer network;

the system being configured or designed to receive file information from selected peer devices, the file information relating to shared files stored at each of the selected peer devices;

the file information including a filename for each shared file, and including a HASH ID for each shared file;

the system being further configured or designed to storing the file information in at least one data structure at the database system; and

the system being further configured or designed to identify a desired shared file in the network using its associated HASH ID.

53. The system of claim 52 being further configured or designed to identify an identity of a peer device using a selected HASH ID;

wherein the identified peer device has been identified as storing a file having an associated HASH ID which matches the selected HASH ID.

54. The system of claim 52 being further configured or designed to identify a network address of a first peer device using a selected HASH ID;

wherein the first peer device has been identified as storing a file having an associated HASH ID which matches the selected HASH ID.

55. A system for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the system comprising:

at least one CPU

memory

at least one interface for communicating with other devices in the peer-to-peer network;

the system being configured or designed to receive a first message from a first peer device, the first message including a search request for locating files in the network which match a first search string;

the system being further configured or designed to generate a first response to the first message, the response including a first list of file records relating to files stored in the network which match the first search string, wherein each file record includes an associated HASH ID and an associated filename; and

the system being further configured or designed to provide the first list of file records to the first peer device.

56. The system of claim 55 further being further configured or designed to exclude from the first list of file records duplicate records in which multiple file records have the same associated HASH ID and filename.

57. The system of claim 55 being further configured or designed to receive a second message from the first peer device in response to the first response, the second message including at least one HASH ID;

the system being further configured or designed to identify, using said at least one HASH ID, at least one network address corresponding to at least one peer device which has been identified as storing at least one file corresponding to the at least one HASH ID; and

the system being further configured or designed to provide, to the first peer device, a second response, the second response including address information which includes at least a portion of the at least one identified network addresses.

58. A system for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the system comprising:

at least one CPU

memory

at least one interface for communicating with other devices in the peer-to-peer network;

the system being configured or designed to identify a first network addresses corresponding to a first peer device which has been identified as storing a first information file associated with a first HASH ID;

the system being further configured or designed to identify a second network addresses corresponding to a second peer device which has been identified as storing a second information file associated with the first HASH ID;

the system being further configured or designed to transmit a first message to the first peer device request a first portion of file content of the first information file from the first peer device; and

the system being further configured or designed to transmit a second message to the second peer device request a second portion of file content of the second information file from the second peer device.

59. The system of claim 58 wherein the first and second messages each include the first HASH ID.

60. The system of claim 58 wherein the first and second messages are initiated at substantially a same time

61. The system of claim 58 wherein the request of the first portion of file content from the first peer device occurs concurrently with the request of the second portion of file content from the second peer device.

62. The system of claim 58 being further configured or designed to receive the first portion of file content from the first peer device;

the system being further configured or designed to receive the second portion of file content from the second peer device; and

the system being further configured or designed to generate a third information file which includes the first and second portion of file content, wherein the file content of the third information file is identical to the file content of the first information file.

63. The system of claim 58 being further configured or designed to detect a failure in a file transfer process associated with the first peer device;

the system being further configured or designed to identify a third network addresses corresponding to a third peer device which has been identified as storing a third information file associated with the first HASH ID;

the system being further configured or designed to transmit a third message to the third peer device, the third message corresponding to a request to retrieve the first portion of file content from the third information file.

64. The system of claim 58 wherein the first portion of file content corresponds to a first chunk of bytes **1** to N of the first information file; and

wherein the second portion of file content corresponds to a second chunk of bytes N+1 to 2N of the second information file.

65. A system for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the system comprising:

at least one CPU

memory

at least one interface for communicating with other devices in the peer-to-peer network;

the system being configured or designed to request from a first plurality of peer devices a respective portion of file content from a respective information file, each respective information file being identified as having identical file content and having an identical first HASH ID being associated therewith.

66. The system of claim 65 being further configured or designed to receive from at least a portion of the first plurality of peer devices respective portions of file content from the respective information file; and

the system being further configured or designed to reconstruct the respective portions of file content to assemble a requested information file having file content identical to that corresponding to the first HASH ID being associated therewith.

67. The system of claim 65 being further configured or designed to create, before request a respective portion, a content map of the file content associated with the first HASH ID, said content map parceling the file content into respective portions from 1 to M.

68. The system of claim 67 being further configured or designed to assign at least one respective portion, from 1 to M, to a first peer device of the first plurality of peer devices to request retrieval thereof.

69. The system of claim 68 being further configured or designed to receive from the first peer device the one respective portion, from 1 to M, of file content from the respective information file; and

the system being further configured or designed to update, upon retrieval of the entire one respective portion from the first peer device, the content map corresponding to the retrieval thereof.

70. The system of claim 69 being further configured or designed to reconstruct, upon retrieval of all respective portions, from 1 to M, of file content, the respective portions to assemble a requested information file having file content identical to that corresponding to the first HASH ID being associated therewith.

71. The system of claim 65 being further configured or designed to identify the network addresses corresponding a first plurality of peer devices, from 1 to X, each of the first plurality of peer devices being identified as storing a respective information file, each having identical file content and having an identical first HASH ID being associated therewith;

72. The system of claim 71 being further configured or designed to create, before request a respective portion, a content map of the file content associated with the first HASH ID, said content map parceling the file content into respective portions from 1 to M, where M>X.

73. A computer program product for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, the computer program product comprising:

a computer usable medium having computer readable code embodied therein, the computer readable code comprising:

computer code for selecting a first information file;

computer code for generating, using fingerprinting algorithm, a first fingerprint ID relating to the content of the first information file; and

computer code for identifying the first information file using the first fingerprint ID.

74. The computer program product of claim 73 wherein the fingerprinting algorithm corresponds to an MD5 Message-Digest algorithm.

75. The computer program product of claim 73 wherein the fingerprinting algorithm corresponds to a Secure Hash Algorithm (SHA1).

76. The computer program product of claim 73 wherein the first information file is stored at a first peer device, and wherein the first information file has an associated first filename, the computer program product comprising:

computer code for storing the first filename and first fingerprint ID at the first peer device.

77. The computer program product of claim 76 further comprising:

computer code for transmitting the first filename and the first fingerprint ID to the database system for storage therein.

78. The computer program product of claim 77 wherein the database system corresponds to a remote database system.

79. The computer program product of claim 73 further comprising:

computer code for selecting a second information file having content identical to the first information file;

computer code for applying the fingerprinting algorithm to the content of the second information file to thereby generate an identical first fingerprint ID to that of the first information file; and

computer code for identifying both the first and the second information file using the first unique fingerprint ID.

80. The computer program product of claim 79 wherein the first information file is stored at a first peer device, and has a first associated filename, and wherein the second information file is stored at a second peer device, and has a has second associated filename, the computer program product further comprising:

computer code for storing the first associated filename and first fingerprint ID associated with the first information file in the database system; and

computer code for storing the second associated filename and first fingerprint ID associated with the second information file in the database system.

81. A computer program product for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective fingerprint ID associated therewith relating to its file content, the computer program product comprising:

a computer usable medium having computer readable code embodied therein, the computer readable code comprising:

computer code for transmitting a first message to the database system, the first message including a search request for locating files in the network which match a first search string; and

computer code for receiving a first response from the database system, the first response including first information relating to identified files stored in the network which match the first search string;

the first information further including an associated fingerprint ID for each identified file.

82. A computer program product for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective fingerprint ID associated therewith relating to its file content, the computer program product comprising:

a computer usable medium having computer readable code embodied therein, the computer readable code comprising:

computer code for transmitting a first message to a first peer device, the first message corresponding to a request to retrieve a first file identified by a first fingerprint ID, wherein the first message includes the first fingerprint ID, and wherein the first fingerprint ID is different than a filename associated with the first file; and

computer code for receiving a first portion of the file content of the first file from the first peer device in response to the first message.

83. A computer program product for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the computer program product comprising:

a computer usable medium having computer readable code embodied therein, the computer readable code comprising:

computer code for receiving file information from selected peer devices, the file information relating to shared files stored at each of the selected peer devices;

the file information including a filename for each shared file, and including a HASH ID for each shared file;

computer code for storing the file information in at least one data structure at the database system; and

computer code for identifying a desired shared file in the network using its associated HASH ID.

84. A computer program product for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the computer program product comprising:

a computer usable medium having computer readable code embodied therein, the computer readable code comprising:

computer code for receiving a first message from a first peer device, the first message including a search request for locating files in the network which match a first search string;

computer code for generating a first response to the first message, the response including a first list of file records relating to files stored in the network which match the first search string, wherein each file record includes an associated HASH ID and an associated filename; and

computer code for providing the first list of file records to the first peer device.

85. A computer program product for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the computer program product comprising:

a computer usable medium having computer readable code embodied therein, the computer readable code comprising:

computer code for identifying a first network addresses corresponding to a first peer device which has been identified as storing a first information file associated with a first HASH ID;

computer code for identifying a second network addresses corresponding to a second peer device which has been identified as storing a second information file associated with the first HASH ID;

computer code for transmitting a first message to the first peer device requesting a first portion of file content of the first information file from the first peer device; and

computer code for transmitting a second message to the second peer device requesting a second portion of file content of the second information file from the second peer device.

86. A system for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the system comprising:

means for identifying a first network addresses corresponding to a first peer device which has been identified as storing a first information file associated with a first HASH ID;

means for identifying a second network addresses corresponding to a second peer device which has been identified as storing a second information file associated with the first HASH ID;

means for transmitting a first message to the first peer device requesting a first portion of file content of the first information file from the first peer device; and

means for transmitting a second message to the second peer device requesting a second portion of file content of the second information file from the second peer device.

87. A system for accessing information in a peer-to-peer network, the peer-to-peer network including a plurality of peer devices and a database system accessible by at least a portion of the peer devices, each of the peer devices being configured to store information files, and further being configured to share content from selected information files with at least a portion of the other peer devices in the network, wherein each shared file in the network has a respective HASH ID associated therewith relating to its file content, the HASH ID being different from a respective filename associated with each file, the system comprising:

means for receiving a first message from a first peer device, the first message including a search request for locating files in the network which match a first search string;

means for generating a first response to the first message, the response including a first list of file records relating to files stored in the network which match the first search string, wherein each file record includes an associated HASH ID and an associated filename; and

means for providing the first list of file records to the first peer device.

* * * * *