IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

EQUIL IP HOLDINGS LLC,	)	
	)	
Plaintiff,	)	C.A. No. 22-677-RGA
	)	
v.	)	**JURY TRIAL DEMANDED**
	)	
AKAMAI TECHNOLOGIES, INC.,	)	
	)	
Defendant.	)	

FIRST AMENDED COMPLAINT FOR PATENT INFRINGEMENT

Pursuant to Federal Rule of Civil Procedure 15(a)(1)(B), Plaintiff Equil IP Holdings LLC ("Equil IP") files this First Amended Complaint for Patent Infringement against Akamai Technologies, Inc. ("Akamai"). This First Amended Complaint supersedes the initial complaint filed on May 24, 2022 (D.I. 1). Equil IP states the following:

NATURE OF THIS ACTION

1. This is a civil action for the infringement of the following United States patents (the "Asserted Patents") as described herein:

- U.S. Patent No. 8,495,242 ("Automated media delivery system"), issued on July 23, 2013 (the "'242 Patent");
- U.S. Patent No. 9,158,745 ("Optimization of media content using generated intermediate media content"), issued on October 13, 2015 (the "'745 Patent"); and
- U.S. Patent No. 6,792,575 ("Automated processing and delivery of media to web servers"), issued on September 14, 2004 (the "'575 Patent").

2. The Asserted Patents are members of a family that also includes U.S. Patent No. 6,964,009 ("Automated media delivery system"), issued on November 8, 2005 (the "'009 Patent"), and U.S. Patent No. 8,381,110 ("Automated media delivery system"), issued on

February 19, 2013 (the "'110 Patent") (collectively with the Asserted Patents, the "Equilibrium Patents").

3. The Equilibrium Patents disclose revolutionary innovations regarding how rich-media content such as images and videos can be optimized and rapidly delivered to Internet-connected devices such as phones and computers.

PARTIES

4. Plaintiff Equil IP is a limited liability company duly organized and existing under the laws of Delaware having its principal place of business at 500 Tamal Plaza, Suite 528, Corte Madera, CA 94925. Equil IP is the owner by assignment of the Equilibrium Patents.

5. With respect to the Equilibrium Patents, Equil IP is the successor-in-interest to Automated Media Processing Solutions, Inc. d/b/a Equilibrium ("Equilibrium"), a corporation duly organized and existing under the laws of Delaware having its principal place of business at 500 Tamal Plaza, Suite 528, Corte Madera, CA 94925.

6. Equilibrium is an industry pioneer and leader in developing patented automated media processing solutions to help its customers manage, modify, and efficiently deploy media-rich content such as images, video, and sound optimized for delivery over the Internet and customized for use on Internet-connected end-user devices such as desktop and laptop computers and mobile phones.

7. On information and belief, Akamai is a corporation organized and existing under the laws of the State of Delaware, with its corporate headquarters and a principal place of business at 145 Broadway, Cambridge, MA 02142.

JURISDICTION AND VENUE

8. This is an action for patent infringement arising under the Patent Act, 35 U.S.C. § 1 *et seq*. Accordingly, this Court has subject-matter jurisdiction under 28 U.S.C. §§ 1331 and 1338(a).

9. This Court has personal jurisdiction over Akamai, as Delaware is Akamai's state of incorporation, and Akamai therefore resides in Delaware. Akamai has also derived revenue from its infringing acts within this District.

10. Venue is proper in this District under 28 U.S.C. §§ 1391(b) and 1400(b), as Delaware is Akamai's state of incorporation, and Akamai therefore resides in Delaware.

THE EQUILIBRIUM PATENTS AND "MEDIARICH SERVER"

11. The technology at issue in this suit traces its roots to the early 1990s when Equilibrium, founded by Sean Barger,[1] acquired and further developed a software product called "DeBabelizer." Sold starting in the early 1990s to industry power-users and consumers alike, DeBabelizer enabled users to automatically edit ("batch process") collections of graphic images, animation, and video across over fifty different file types. It was powerful software, the full version of which sold for hundreds of dollars. The name "DeBabelizer" referred to the program's ability to import and normalize any format, then systematically edit a wide variety of otherwise incompatible media file types and then export them to any format automatically. After selling over 1 million copies of DeBabelizer Lite and DeBabelizer Pro for Macintosh, Windows OS's and Silicon Graphics, Equilibrium believed a logical expansion of the franchise was to develop a way to deliver high-speed, dynamic content through a system tied directly into web servers. The

[1] Equilibrium was originally formed to develop computer games.

team set out to build a next-generation technology which solved the automatic rendition management problem to enable Web 2.0 for scaling any size website while providing unique device support without having to pre-process content for all viewing scenarios. They called the project "Freeride."

12. With the growth of the web in the late 1990s, Equilibrium foresaw the Freeride project developing a new kind of Internet-based service that web developers could use to simplify the process of optimizing and maintaining rich media content automatically on websites. Equilibrium envisioned a system that would enable next-generation ecommerce and other applications.

13. On October 21, 1999, in order to protect this system, Equilibrium filed the first in a series of U.S. patent applications that would ultimately issue as the Equilibrium Patents. All the Equilibrium Patents are members of the same family and share substantially similar disclosures.

14. That first application (Application No. 09/425,326) issued as the '575 Patent on September 14, 2004. A copy of the '575 Patent is attached as Exhibit A.

15. On August 16, 2000, Equilibrium filed Provisional Application No. 60/226,043, and on August 14, 2001, it filed Continuation-in-Part Application No. 09/929,904, which issued as the '009 Patent on November 8, 2005. A copy of the '009 Patent is attached as Exhibit B.

16. On September 26, 2008, Equilibrium filed Divisional Application No. 12/238,842, which issued as the '110 Patent on February 19, 2013. A copy of the '110 Patent is attached as Exhibit C.

17. On February 26, 2010, Equilibrium filed Divisional Application No. 12/713,637, which issued as the ’242 Patent on July 23, 2013. A copy of the ’242 Patent is attached as Exhibit D.

18. On July 15, 2008, Equilibrium filed Divisional Application No. 12/173,747, which issued as U.S. Patent No. 8,656,046 on February 18, 2014. A copy of the ’046 Patent is attached as Exhibit E.

19. On January 28, 2013, Equilibrium filed Continuation Application No. 13/752,110, which issued as the ’745 Patent on October 13, 2015. A copy of the ’745 Patent is attached as Exhibit F.

20. On December 17, 2021, Equilibrium assigned the Equilibrium Patents to Equil IP Holdings LLC.

21. The Equilibrium Patents individually and collectively disclose substantial improvements to then-existing systems of Internet media delivery. Existing systems did not enable on-the-fly rich media generation and caching of rich media that was optimized for each user. In the novel methods described in the Equilibrium Patents, each user’s request contains information identifying the requested media and further indicating incremental optimizations that are performed prior to delivery to the user. The incrementally optimized media is cached for future delivery if another user request is received containing information requesting the same media and indicating the same incremental optimizations.

22. The technology disclosed in the Equilibrium Patents is widely used today to power the intelligent edge networks that content providers rely on to deliver media-rich Internet web pages and videos quickly and with very high quality. Without the innovations of the Equilibrium Patents, media-rich Internet content would be much slower to deliver and of lower

quality and/or would require expensive investments in computing and network infrastructure. The Equilibrium Patents therefore disclose substantial improvements to computer and network technology. Those improvements were not well-understood, routine, or conventional in the "Web 1.0" computing and network environment that existed at the time they were filed.

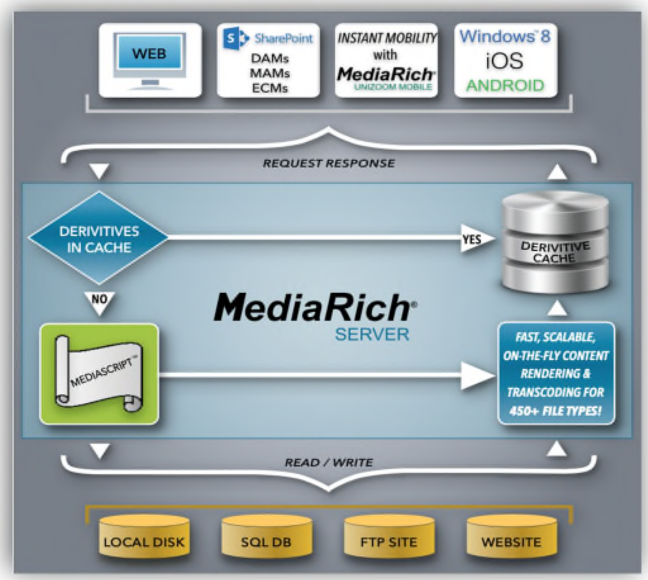
23. The Equilibrium Patents have been widely cited, including by Akamai. According to Google Patents, the '575 Patent has been cited more than 200 times by other U.S. patents and patent applications. Among those citations to the Equilibrium Patents are more than twenty patents and patent applications — from multiple patent families directed to numerous aspects of a content delivery network — that are identified as assigned to Akamai. (*See, e.g.*, U.S. Patent Nos. 7,653,706 ("Dynamic image delivery system"); 7,725,602 ("Domain name resolution using a distributed DNS network"); 7,912,978 ("Method for determining metrics of a content delivery and global traffic management network"); 7,996,533 ("HTML delivery from edge-of-network servers in a content delivery network (CDN)"); 8,805,965 ("Methods and apparatus for image delivery"); 9,418,353 ("Methods and systems for delivering content to differentiated client devices"); 9,419,852 ("Systems and methods for identifying and characterizing client devices"); 9,509,804 ("Scalable content delivery network request handling mechanism to support a request processing layer"); 9,544,183 ("Methods and apparatus for providing content delivery instructions to a content server"); 9,654,579 ("Scalable content delivery network request handling mechanism"); 9,742,858 ("Assessment of content delivery services using performance measurements from within an end user client application"); and 9,817,916 ("Methods and apparatus for accelerating content authored for multiple devices").) In several cases, the '575 Patent was the basis for a rejection by an examiner. (*See, e.g.*, U.S. Application Nos. 11/009,273; 11/768,935; and 12/693,413.)

24. The technology disclosed and claimed in the Equilibrium Patents was released by Equilibrium in 2000 as the "MediaRich Server." Since that time, Equilibrium has continuously offered MediaRich Server services in the business-to-business market. Recent clients include Walmart, the Walt Disney Company, Sony Pictures, Warner Bros., The Metropolitan Museum of Art, the U.S. Department of Energy, and E. T. Browne Drug Co. These companies use services powered by Equilibrium technology to automatically process, manage, and serve rich media content to websites and mobile devices.

25. The current version of one of the Equilibrium services that practice inventions disclosed in the Equilibrium Patents is MediaRich 6. (*See, e.g.*, Ex. G ("MediaRich 6 Brochure").)

26. One aspect of the MediaRich Server — and the Equilibrium Patents that claim the technology that powers it — is the ability to optimize rich media on-the-fly and cache what Equilibrium calls media "renditions" in real time in response to Internet user requests. (*See id.* at 3.) As illustrated in the graphic below, users of a wide variety of devices (from web browsers on desktop computers to apps on mobile devices) send requests for media content to websites

and other network-hosted sources. Equilibrium's service sits between those two parts of the network:



27. MediaRich solves technical problems associated with delivering rich media content over the Internet that arise from the fact that each type of user device has different parameters that can be used to optimize media for viewing. Parameters like image size and resolution, compression type, compression level, frame rate, and many others can be used to optimize Internet media to make the most of each user's type of device, specific application, quality, and quality of Internet service available at the moment, and other attributes. The MediaRich Server detects or is sent optimization parameters and optimizes the requested media on the fly before sending it on to the user.

28. Once an optimized rendition is created, moreover, it is cached in the MediaRich Server. The next time any user is determined to have a need or request for the same media, the server checks the cache to see if a previously optimized rendition already exists. If it does, the pre-existing rendition is further optimized, if necessary, and sent. If not, a rendition can be made

from the original media, sent, and cached to be used as intermediate or final media for the next request. Over time, the library of renditions in the cache builds up, further improving the responsiveness of the system and thus improving user experience.

THE AKAMAI "INTELLIGENT EDGE" NETWORK

29. Mr. Barger has been an evangelist for the Equilibrium technology for many years. Equilibrium has partnered with industry-leading Internet companies to enhance their services with Equilibrium technology. Among them is Akamai, a leading Internet platform provider. Unfortunately, Akamai has elected over the years to become an "efficient infringer" by recreating Equilibrium's patented service rather than continuing to partner with Equilibrium or licensing Equilibrium's technology.

30. On information and belief, Akamai was founded in the late 1990s to help solve the growing problem of Internet latency by offering Content Delivery Network (CDN) services. A CDN is a network of computers located at geographically dispersed data centers. The CDN network computers replicate and store website information so that it is available at physical locations closer to users who might access the website. On information and belief, at the time of its founding, Akamai's network replicated website data and distributed it throughout the network to pre-position it closer to users, rather than generate optimized content on the fly.

31. In February 2000, Equilibrium reached out to Akamai to explore options for integrating Equilibrium's MediaRich technology with Akamai's CDN service. Equilibrium explained:

> Equilibrium's next generation product building on our 10 years of automated media processing experience, patent pending, is something that is a natural extension to the Akamai network . . . . It is aimed at content and media servers, integrates with existing server/database environments and dynamically generates media and content to the webserver cache on-the-fly. The system handles images, animation, digital video and sound currently, and has been architected to handle

> any media type . . . . It . . . handles media cache management for web servers and creation/delivery of media to handheld/wireless devices automatically so that the web designer doesn't have to integrate with proxy server systems or create multiple sites.

32. Akamai's then Chief Operating Officer (and future Chief Executive Officer) Paul Segan responded, directing his staff to reach out to Equilibrium "to learn more about Equilibrium and to assess how Akamai might become involved with [Equilibrium]."

33. Mr. Barger and others from Equilibrium met with three Akamai Director-level executives and its principal engineer four days later. Contemporaneous memoranda indicate that Akamai "saw a clear synergy between Equilibrium's media serving technology and Akamai's data distribution system and expressed interest in further exploring possibilities for integration of the two technologies."

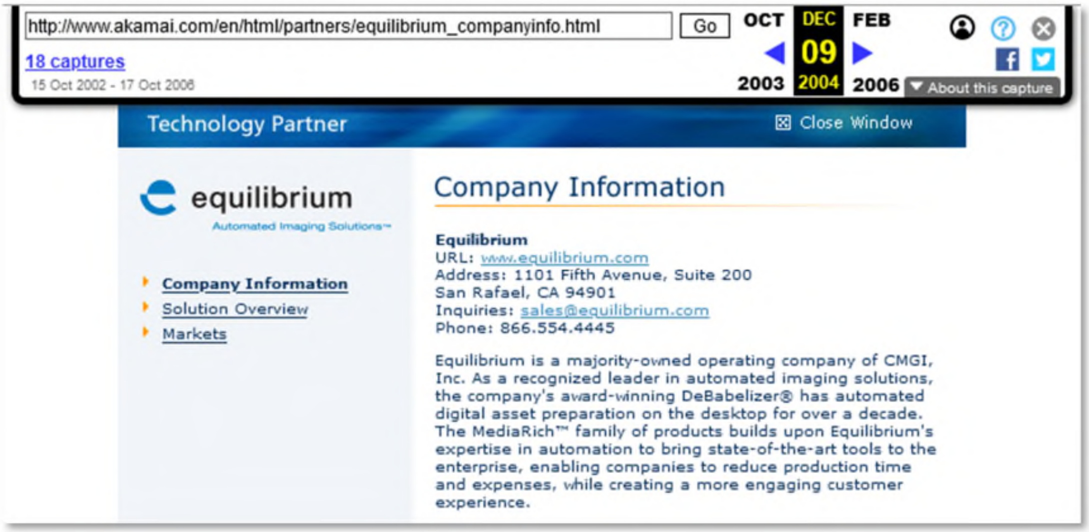
34. Akamai's Principal Engineer James Kistler expressed skepticism about whether original media could be brought to the edge of the network for dynamic image generation. Akamai's Director of Business Development Ravi Sundararajan suggested that the Equilibrium technology could be a good fit for a new initiative Akamai was then architecting called "EdgeAdvantage." In this new system, Akamai suggested that Equilibrium's media server could connect directly to Akamai's caching servers closer to the edge of the network. Mr. Barger agreed that this would be a good approach.

35. The parties also agreed to include Equilibrium in Akamai's "Technology Partner Program." The Akamai website described the "Technology Partner Program" as follows:

> Akamai Technology Partners work with Akamai to utilize Internet edge technology to create feature-enhanced products that enable customers to quickly and easily leverage Akamai's e-business infrastructure services and provide joint, industry-leading solutions for distributed content and application delivery.

Other partners included IBM, Microsoft, and Oracle. Equilibrium's inclusion in this program made it easy for Akamai's customers to use Equilibrium MediaRich services through the Akamai Content Delivery Network.

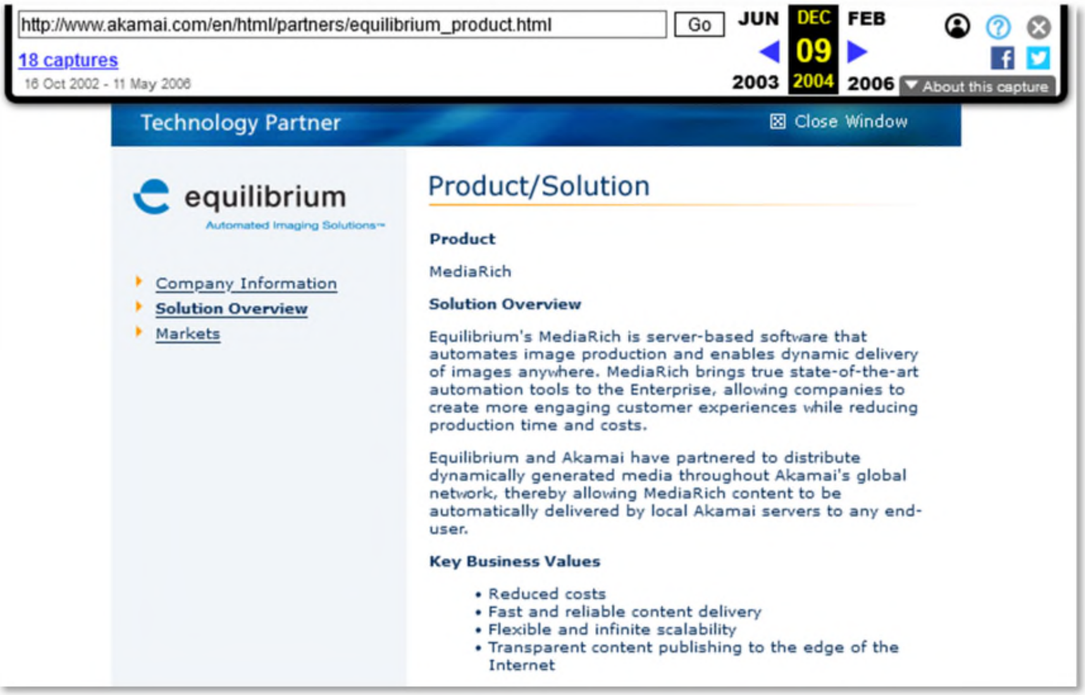
36. Equilibrium's inclusion in the Akamai "Technology Partner Program" was documented on the Akamai website, including, for example, in these December 9, 2004, Internet Archive captures:



Company Information

Equilibrium
URL: www.equilibrium.com
Address: 1101 Fifth Avenue, Suite 200
San Rafael, CA 94901
Inquiries: sales@equilibrium.com
Phone: 866.554.4445

Equilibrium is a majority-owned operating company of CMGI, Inc. As a recognized leader in automated imaging solutions, the company's award-winning DeBabelizer® has automated digital asset preparation on the desktop for over a decade. The MediaRich™ family of products builds upon Equilibrium's expertise in automation to bring state-of-the-art tools to the enterprise, enabling companies to reduce production time and expenses, while creating a more engaging customer experience.



Product/Solution

Product

MediaRich

Solution Overview

Equilibrium's MediaRich is server-based software that automates image production and enables dynamic delivery of images anywhere. MediaRich brings true state-of-the-art automation tools to the Enterprise, allowing companies to create more engaging customer experiences while reducing production time and costs.

Equilibrium and Akamai have partnered to distribute dynamically generated media throughout Akamai's global network, thereby allowing MediaRich content to be automatically delivered by local Akamai servers to any end-user.

Key Business Values

- Reduced costs
- Fast and reliable content delivery
- Flexible and infinite scalability
- Transparent content publishing to the edge of the Internet

Akamai stated that Equilibrium was a "recognized leader in automated imaging solutions." Akamai further stated: "Equilibrium's MediaRich is server-based software that automates image production and enables dynamic delivery of images anywhere. MediaRich brings true state-of-the-art automation tools to the Enterprise, allowing companies to create more engaging customer experiences while reducing production time and costs."

37. However, Akamai resisted Equilibrium's offers to establish a more extensive partnership or more complete integration of Equilibrium's technology into Akamai's edge network.

38. Over the years, from time-to-time, Equilibrium and Akamai would discuss whether an integrated product offering would be appropriate. For example, when John Bishop joined Akamai from Cisco in 2014 to become Akamai's Chief Technology Officer, Mr. Barger met with him at an industry conference in Las Vegas. The pair later exchanged messages. Mr. Barger proposed that they should discuss opportunities for further integration of the two

companies' products. Mr. Bishop told Mr. Barger that he was just settling in and wanted to get the "lay of the land" before continuing discussions. The discussions never resumed, however.

39. A little over two years later, Akamai announced it had developed a "new image management service" that it called "Image Manager." Just like Equilibrium's MediaRich service, Akamai's Image Manager promised to "engage online audiences with attractive images that are automatically optimized for both maximum visual quality and performance while reducing the cost and effort to transform and deliver web-ready images." Image Manager promised "[f]aster sites and apps on any device, anywhere by reducing bytes delivered to web and mobile users."

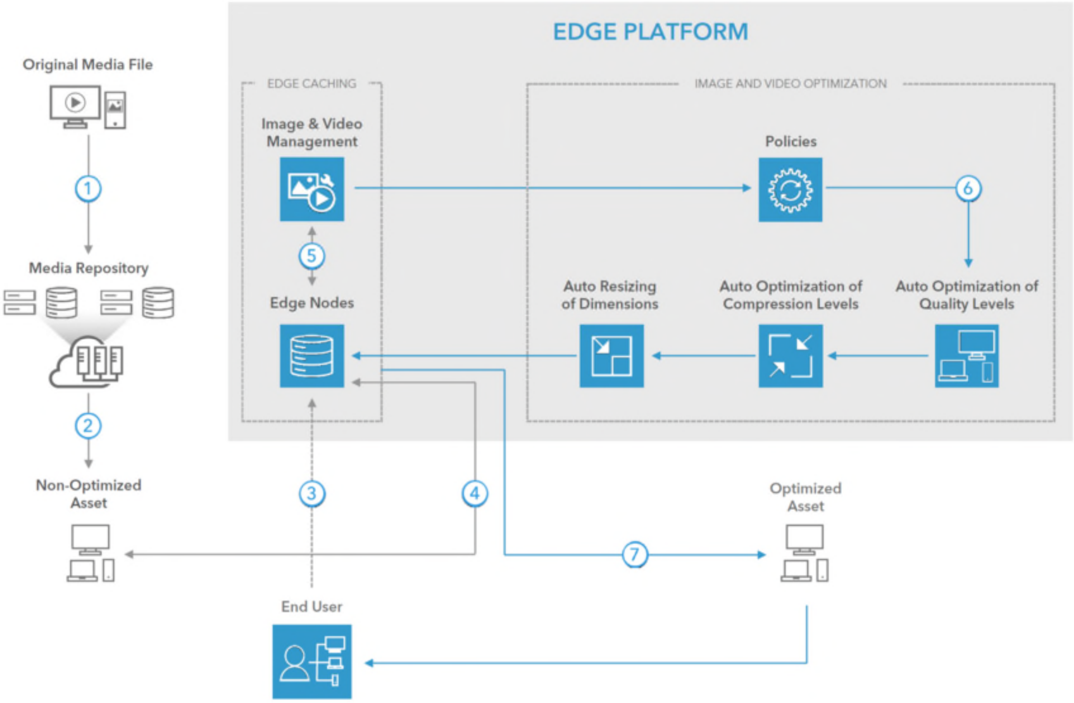
40. Image Manager was one of Akamai's "key offerings." In its 2016 Annual Report, in a section entitled "Web and Mobile Performance Solutions," Akamai stated:

> The ultimate goal of our web and mobile performance business is to make dynamic websites and applications have instant response times, no matter where the user is, what device or browser they are using, or how they are connected to the Internet. This is accomplished through a variety of advanced technologies embedded into our platform, which can be thought of as a virtual Internet overlaying the real Internet.

The report then identified several "key offerings" of its "virtual Internet," including: "Image Manager – To help our customers cope with the multitude of devices used by their consumers and varying connection quality, Image Manager automatically optimizes online images for the best combination of size, quality, and file format suited for each image and device and offloads the artistic transformation of derivative assets to the cloud."

41. Just like MediaRich, Akamai's Image Manager (today called "Image and Video Manager" or "IVM") stores "one high-quality, pristine file" in a media repository as a "Non-Optimized" Asset. Data from the asset is transmitted to "Edge Nodes" of an "Edge Cach[e]." The cached data is transmitted for "Image and Video Optimization" that automatically adjusts

factors such as quality, compression, and dimension resizing, based on information received about the "user context, including browser, device types, and the quality level of the end user's current network connection." The optimized media is cached at the Edge Node and served as an "Optimized Asset" to end users.



42. According to Akamai:

> [IVM] means developers no longer need to worry about setting image-level quality. By automatically optimizing images for maximum perceived quality with minimum image weight, the algorithm lets developers relax knowing that images will be delivered at a specified quality that's optimal for end users.

43. As a result, Akamai boasts that rich media content is "[o]ptimized for every visitor" and "simplified for developers."

44. IVM is just one example of an infringing Akamai service that replicates the features of Equilibrium's MediaRich Server service. Other features of services that grew out of Akamai's "EdgeAdvantage" to be part of Akamai's "Intelligent Edge Platform" employ image

and video optimization methodologies infringing one or more of the method claims of the Equilibrium Patents. For the avoidance of doubt, Equil IP asserts that Akamai infringes only method claims of the Equilibrium Patents.

EQUILIBRIUM'S PATENTED INVENTIONS

45. In this First Amended Complaint for Patent Infringement, Equil IP asserts three related patents. Generally speaking, the '745 and '242 Patents claim methods by which rich media content is optimized and delivered to users in response to user requests. The '575 Patent, which was the first in the family to issue, relates to the process by which the rich media optimization system interacts with the HTML or other browser language code in which a website containing rich media is written and which the user's browser decodes when viewing the website.

46. The '745 Patent is titled "Optimization of Media Content Using Generated Intermediate Media Content" and it issued on October 13, 2015. It claims priority to an application filed on October 21, 1999.

47. The '745 Patent claims an "automated graphics delivery system." (Ex. F ('745 Patent) at 5:1-5.) The disclosed invention improves on prior art web media delivery systems by providing media "[a]sset management, automatic image manipulation, automatic image conversion, automatic image upload, and automatic disk management." (*Id.* at 8:8-13.) The automatic image optimization, delivery, and caching is done on demand and in real time in response to user requests. "Web media is generated only if requested by a client browser." (*Id.* at 5:35.) This results in a "dynamic Web site wherein images are generated on demand from original assets, wherein only the original assets need to be updated." (*Id.* at 5:11-14.)

48. Because images are optimized on demand in response to specific user requests, the media can be optimized to fit the specific needs of the individual user. For example, the system can optimize media throughput based on "current Web server traffic." (*Id.* at 5:15-17.) Similarly, the system can factor "client connection speed" to determine "optimum quality and file size." (*Id.* at 5:19-20.)

49. In the Equilibrium System, once media has been optimized in response to a user request, it is stored in a cache. A "media creation subsystem" is employed to create the optimized image and then "store the results in the media cache database." (*Id.* at 8:57-60.) Since this optimized media is automatically cached, "identical requests can be handled without regeneration of images." (*Id.* at 5:24-25.)

MARKING

50. Equilibrium was not required to mark under 35 U.S.C. § 287 because its MediaRich service is a software system implementing patented methods that lacks a tangible item to mark.

51. Additionally, claim 1 of the '745 Patent, claim 9 of the '242 Patent, and claim 1 of the '575 Patent are method claims which do not give rise to an obligation to mark under 35 U.S.C. § 287.

THE ACCUSED PRODUCTS

52. On information and belief, Akamai offers a variety of Internet content delivery services that it packages and repackages under a variety of trade names. Many of these are offered as part of what it calls the "Akamai Intelligent Edge." On information and belief, Akamai pervasively uses Equilibrium's patented inventions in its Intelligent Edge and related services. A few specific examples are set forth below.

Image and Video Manager

53. Akamai's Image and Video Manager feature of its Intelligent Edge is one example of how Akamai's services infringe. IVM practices all the limitations of one or more of the Equilibrium Patent method claims, including at least exemplary independent claim 1 of the '745 Patent.

54. Claim 1 of the '745 Patent recites:

> [Preamble] A method in a host computer for developing transformation processing operations to optimize media content playback to a plurality of playback devices connected with the host computer in a network, the method comprising:
>
> [1a.] receiving a first request from a first playback device for media content;
>
> [1b.] wherein the first request contains information, the information indicating a first original media content, first content generation operations, and first transformation operations;
>
> [1c.] determining whether a previously-generated first intermediate media content is available for reuse, the previously-generated first intermediate media content having been created using the first original media content and the first set of content generation operations; and
>
> [1d.] responsive to determining that a previously-generated first intermediate media content is available, creating a first optimized media content for the first playback device by performing the first set of transformation operations on the previously-generated first intermediate media content; and
>
> [1e.] responsive to determining that a previously-generated first intermediate media content is not available, creating a first optimized media content for the first playback device by creating a first intermediate content using the first original media content and the first set of content generation operations, and performing the first set of transformation operations on the first intermediate media content; and
>
> [1f.] sending the first optimized media content to the first playback device.

(Ex. F at 23:9-39.)

55. Akamai's Image and Video Manager employs "a method in a host computer for developing transformation processing operations to optimize media content playback to a plurality of playback devices connected with the host computer in a network." (*Id.* at 23:9-12.) It employs a method in Akamai host edge network computers that develops transformation processing operations to optimize media content playback to a plurality of playback devices, such as computers and mobile phones, which are connected to the Akamai edge network via the Internet. Thus, the preamble, if limiting, is satisfied.

56. According to Akamai, "Image & Video Manager is a software as a service (SaaS) solution that automatically optimizes and enhances images and videos for every user on the fly. It provides each user the ideal combination of quality, format, and size that is best suited for that user's browser, device, and network connection at the very moment they access a website or mobile app." (Ex. H ("Reference Architecture") at Overview.)

57. Akamai publishes the following "Reference Architecture" to describe the methods employed in IVM:

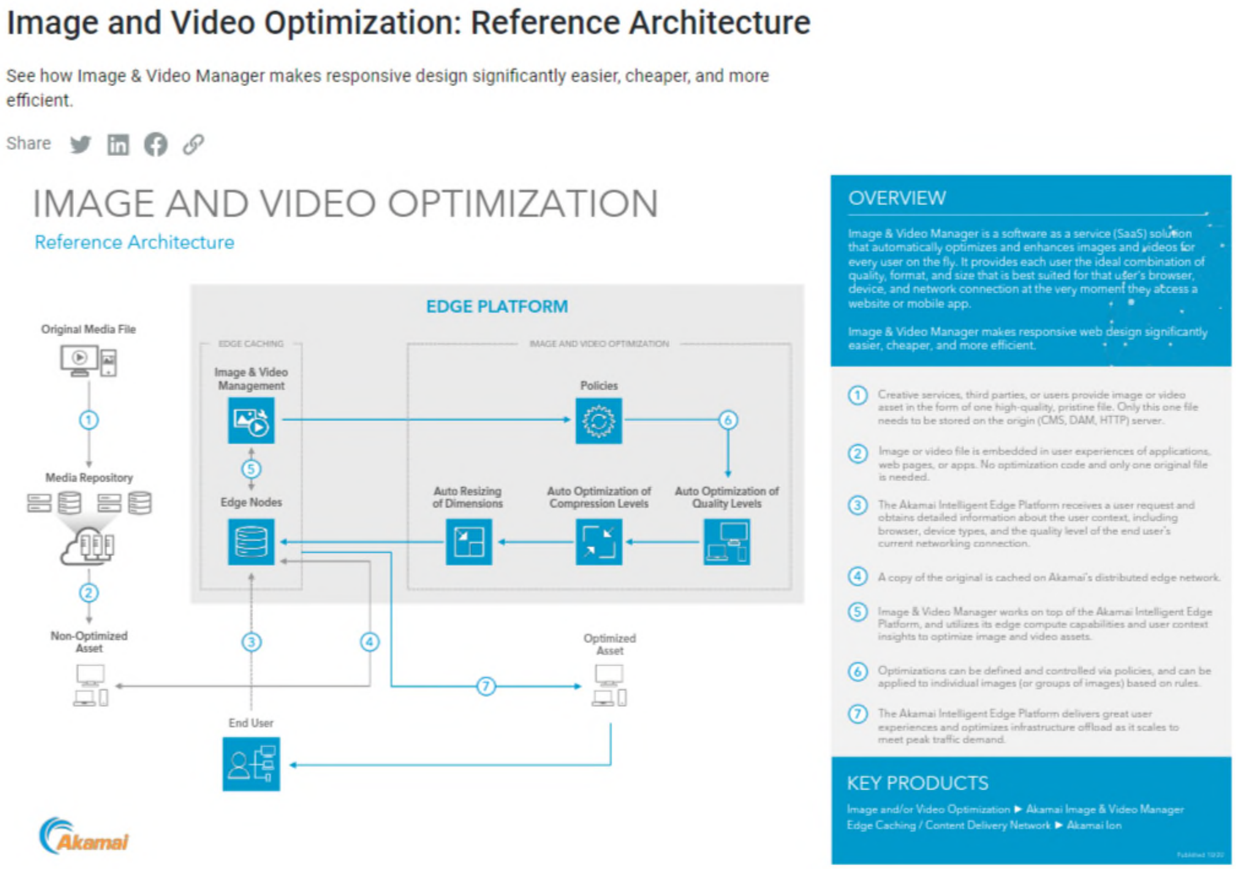


Image and Video Optimization: Reference Architecture

See how Image & Video Manager makes responsive design significantly easier, cheaper, and more efficient.

Share

IMAGE AND VIDEO OPTIMIZATION

Reference Architecture

OVERVIEW

Image & Video Manager is a software as a service (SaaS) solution that automatically optimizes and enhances images and videos for every user on the fly. It provides each user the ideal combination of quality, format, and size that is best suited for that user's browser, device, and network connection at the very moment they access a website or mobile app.

Image & Video Manager makes responsive web design significantly easier, cheaper, and more efficient.

1 Creative services, third parties, or users provide image or video asset in the form of one high-quality, pristine file. Only this one file needs to be stored on the origin (CMS, DAM, HTTP) server.

2 Image or video file is embedded in user experiences of applications, web pages, or apps. No optimization code and only one original file is needed.

3 The Akamai Intelligent Edge Platform receives a user request and obtains detailed information about the user context, including browser, device types, and the quality level of the end user's current networking connection.

4 A copy of the original is cached on Akamai's distributed edge network.

5 Image & Video Manager works on top of the Akamai Intelligent Edge Platform, and utilizes its edge compute capabilities and user context insights to optimize image and video assets.

6 Optimizations can be defined and controlled via policies, and can be applied to individual images (or groups of images) based on rules.

7 The Akamai Intelligent Edge Platform delivers great user experiences and optimizes infrastructure offload as it scales to meet peak traffic demand.

KEY PRODUCTS

Image and/or Video Optimization ▶ Akamai Image & Video Manager
Edge Caching / Content Delivery Network ▶ Akamai Ion

Published 10/20

(*Id.*)

58. According to Akamai, IVM "comes out of the box ready to help developers optimize images, videos, and animated GIFs to deliver great web experiences." (Ex. I (Akamai Image and Video Manager Free Developer Trial Brochure) at 1.)

59. IVM employs a step of "receiving a first request from a first playback device for media content." (Ex. F at 23:13-14.) Step 3 of the Akamai IVM workflow depicted in the Reference Architecture above is: "The Akamai Intelligent Edge Platform receives a user request and obtains detailed information about the user context, including the browser, device types, and the quality level of the end user's current networking connection." (Ex. H, Step 3.) According to Akamai, "all [I]mage [M]anager transformations whether they are image optimizations[,] image transformations or . . . video transformations . . . all happen at the time of the request . . .

so if we don't have the variant video asset inside of cache then that very first request for that video asset would mean that we have to go and that's when we will kick off that very first encoding that occurs." (Ex. J ("Webinar - Image Manager 4.0") at 1.)

60. IVM employs a step "wherein the first request contains information, the information indicating a first original media content, first content generation operations, and first transformation operations." (Ex. F at 23:15-18.) The Akamai IVM workflow depicted in the Reference Architecture above demonstrates that in the IVM system, the workflow is directed to accessing and optimizing a "one high-quality, pristine file" referred to as an "Original Media File." (Ex. H, Step 1.) The first request that IVM receives has information containing, for example, "browser, device types, and the quality level of the end user's current networking connection." (*Id.*, Step 3.) IVM uses this information as indications from which it determines which "image transformations" and "image optimizations" to perform on which original media content. (Ex. J at 4.) The IVM also "automatically optimizes and enhances images and videos for every user on the fly." (Ex. H, Overview.) Thus, IVM provides "each user the ideal combination of quality, format, and size that is best suited for that user's browser, device, and network connection at the very moment they access a website or mobile app" based on the information contained in the first request. (*Id.*)

61. IVM employs a step of "determining whether a previously-generated first intermediate media content is available for reuse, the previously-generated first intermediate media content having been created using the first original media content and the first set of content generation operations." (Ex. F at 23:19-23.) When a user sends a request, Akamai's IVM checks for a cached copy of a previously-generated first intermediate media content. According to Akamai, "whenever the user makes a first request . . . so all [I]mage [M]anager

transformations whether they are image optimizations[,] image transformations or any of the video transformations that we showed all happen at the time of the request so we leverage our caching system pretty heavily to push and to keep all the derivative assets that we create in cache so that we can deliver them out to the end user as quickly as possible and reduce that latency[.] If the assets aren't in the cache then we have to do a transformation and that's what we call the real time or the cache miss hit . . . . [S]o if we don't have the variant video asset inside of cache then that very first request for that video asset would me that we have to go and that's when we will kick off that very first encoding that occurs." (Ex. J at 1-11, 13-17.)

62. Akamai IVM employs a step wherein "responsive to determining that a previously-generated first intermediate media content is available, creating a first optimized media content for the first playback device by performing the first set of transformation operations on the previously-generated first intermediate media content." (Ex. F at 23:24-29.) The IVM architecture uses the "Edge Caching" architecture of Akamai's Intelligent Edge Platform to store previously-requested intermediate media content at the edge of the network. (*See* Ex. H.) A copy of the original is cached on Akamai's distributed edge network. IVM leverages the caching system to "keep all of the derivative assets" that it creates in response to user requests so that it can "deliver them out to the end user as quickly as possible" and achieve reduced latency. (Ex. J at 7-9.) The IVM architecture can automatically optimize "compression" and "quality levels." (Ex. H, Step 6 in diagram.) As a result, according to Akamai, it "provides each user the ideal combination of quality, format, and size that is best suited for that user's browser, device, and network connection at the very moment they access a website or mobile app." (*Id.*, Overview.)

63. Akamai IVM employs a step wherein "responsive to determining that a previously-generated first intermediate media content is not available, creating a first optimized media content for the first playback device by creating a first intermediate content using the first original media content and the first set of content generation operations, and performing the first set of transformation operations on the first intermediate media content." (Ex. F at 23:30-37.) In Akamai's IVM architecture, when the user makes a first request for an original media for which an intermediate media content is not available, that is called a "real time" or "cache miss hit." (Ex. J at 11.) In that case, the IVM pulls and transforms the asset, caches it, and makes it available for optimization and delivery to the end user. (*See* Ex. H, Step 6 in diagram.)

64. IVM employs a step of "sending the first optimized media content to the first playback device." (Ex. F at 23:38-39.) IVM transmits the optimized media content to the requesting computer, mobile phone, or other playback device over the Internet. (*See also* Ex. H, Step 7.)

Adaptive Media Delivery

65. Akamai's Adaptive Media Delivery is another example of how Akamai's Intelligent Edge service employs an infringing method.

66. Claim 9 of the '242 Patent recites:

> [Preamble] A method for accessing dynamically transcoding media content, the method comprising:
>
> [9a.] an act of receiving a request for media content to be delivered to a client presentation system for media content, wherein the requested media content has a limited number of base transcoding profiles associated therewith, each base transcoding profile corresponding to a cached version of the requested media content:
>
> [9b.] at the time of the request, and without input by a network administrator, an act of automatically identifying transcoding parameters to be applied to the requested media content prior to delivery to the client presentation system,

> wherein identification of transcoding parameters is based on one or more formats of any client presentation system;
>
> [9c.] an act of determining that the transcoding parameters to be applied to the requested media content prior to delivery to the client presentation system are the same as transcoding parameters that are being applied to the requested media content prior to delivery to another client presentation system;
>
> [9d.] an act of transcoding the requested media content in accordance with the identified transcoding parameters, such that the identified transcoding parameters are used to perform additional incremental transcoding on top of the base transcoding profile;
>
> [9e.] wherein the act of act of transcoding the requested media content in accordance with the identified transcoding parameters comprises;
>
> [9e(i).] an act of selecting a pre-existing base transcoded version of the requested media content comprising intermediate derivative media that has been transcoded in accordance with only a portion of the identified transcoding parameters; and
>
> [9e(ii).] an act of creating a final version by incrementally performing further transcoding of the pre-existing base transcoded version in accordance with a remaining portion of the identified transcoding parameters; and
>
> [9f.] an act of delivering the transcoded media content to both client presentation systems concurrently.

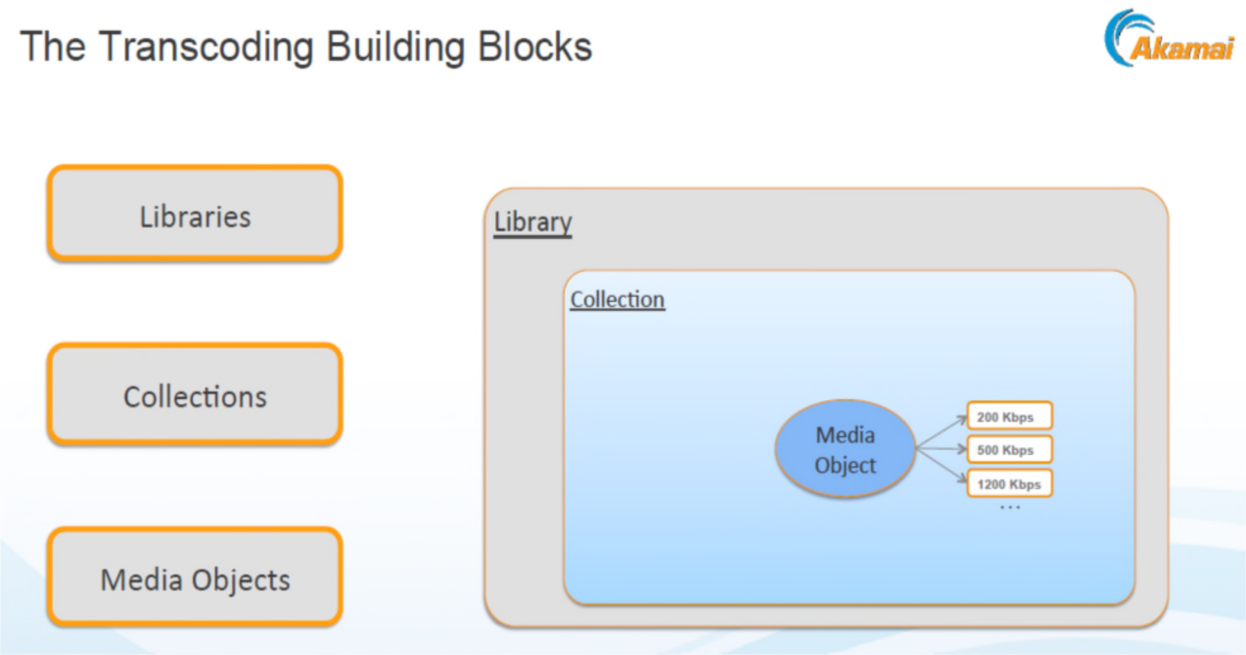
(Ex. D at 23:45-24:16.)

67. Akamai’s Adaptive Media Delivery (“AMD”) employs a computer program that performs all the acts described in claim 9 of the ’242 Patent, and therefore infringes. (*See* Ex. K (adaptive-media-delivery.pdf.)) As discussed in connection with the additional limitations of claim 9, AMD dynamically transcodes media content. Thus, the ’242 Patent preamble, if limiting, is also satisfied.

68. AMD performs an “act of receiving a request for media content to be delivered to a client presentation system, wherein the requested media content has a limited number of base transcoding profiles associated therewith, each base transcoding profile corresponding to a cached version of the requested media content.” (Ex. D at 23:47-52; *see also* Ex. M

(techdocs.akamai.com “Define property hostnames” webpage) at 1, Ex. N (“Akamai edge 2012 Customer Conference” slides) at 10-14.)

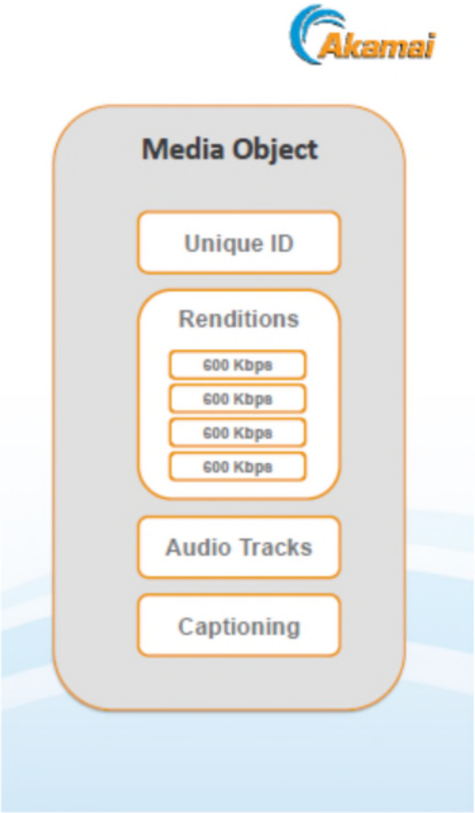
69. According to Akamai, “End-user requests” are directed to and received by “Akamai edge servers” that read media and determine how to deliver it. (Ex. M at 1.) The requested media has associated with it a limited number of base transcoding profiles, each of which corresponds to a cached version of the requested media. (*See* Ex. N at 11, 14.) For example, in the AMD system, a media object in a collection, which must be cached, has different transcoded versions with different bit rates which Akamai depicts as follows:



The Transcoding Building Blocks

(*Id*. at 11.)

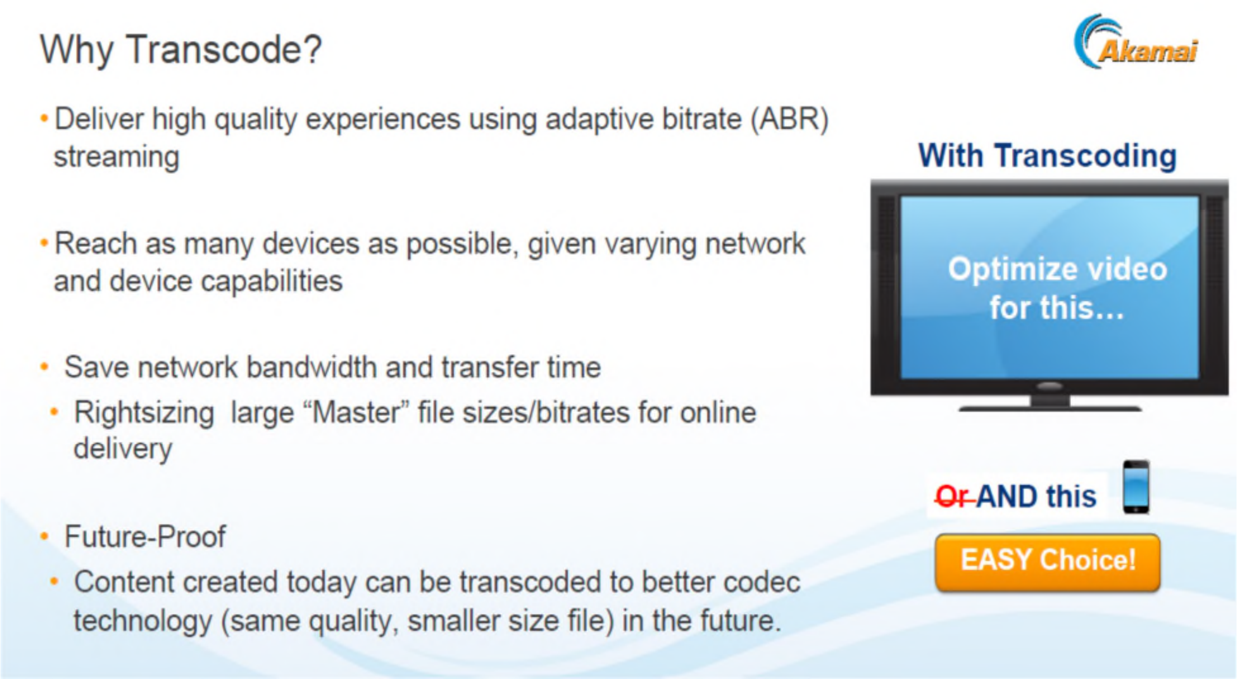
70. Additionally, each media object in AMD may have different renditions, which Akamai depicts as follows:



(*Id.* at 14.) AMD's dynamic transcoding method thus satisfies Element 9a.

71. The methods that AMD employs "identif[y] transcoding parameters to be applied to the requested media content" based on attributes of the end-user device. (Ex. D at 23:53-59; *see also* Ex. N at 2, 5.) It provides live transcoding in real time to different types of end-user devices. (*See* Ex. D at 23:53-59; *see also* Ex. O ("Transcoding 24x7").) It uses "[c]onfiguration elements," such as "rules" and "behaviors," for determining how to respond to requests from end users. (Ex. P (techdocs.akamai.com "Key concepts and terms" webpage) at 4.) These rules and behaviors, among other things, determine how the AMD system responds to requests for media content without input by a network administrator. (*See id.*) The AMD automatically selects and streams the highest quality bit rate to a video end point based on the end-point device "connection speed and device capabilities." (Ex. Q ("Akamai MSOD: Stream Packaging User Guide") at 2.) Each video is personalized for each viewer based on "device type, network

conditions" and other factors. (Ex. K at 4.) Akamai depicts AMD's capabilities graphically as follows in which AMD's "transcoding" enables optimizing rich media for both television and mobile phone device formats:



Why Transcode?

- Deliver high quality experiences using adaptive bitrate (ABR) streaming
- Reach as many devices as possible, given varying network and device capabilities
- Save network bandwidth and transfer time
  - Rightsizing large "Master" file sizes/bitrates for online delivery
- Future-Proof
  - Content created today can be transcoded to better codec technology (same quality, smaller size file) in the future.

With Transcoding

Optimize video for this...

~~Or~~ AND this

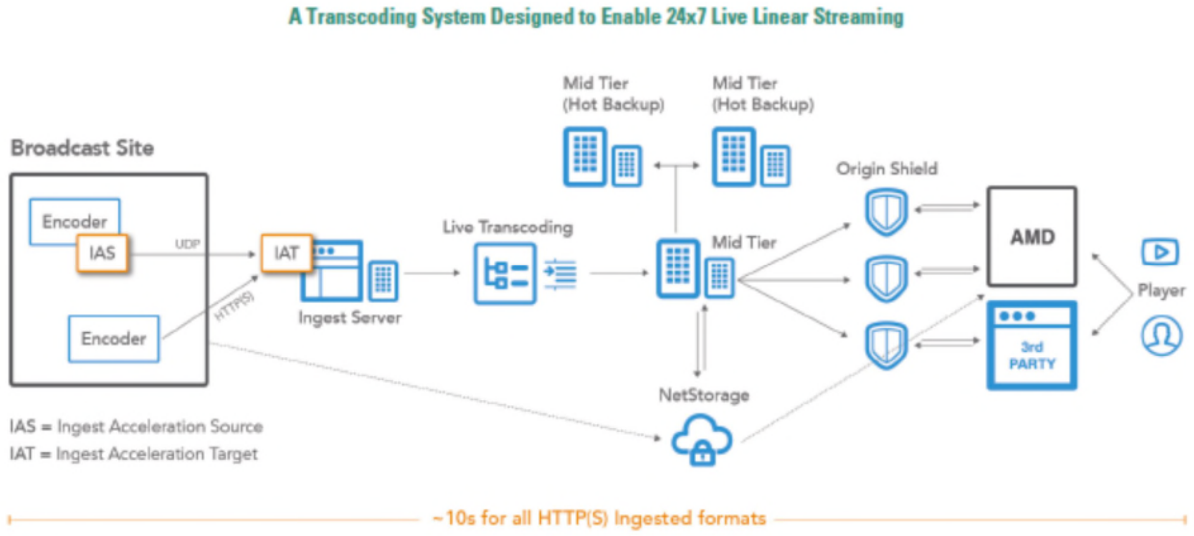
EASY Choice!

(Ex. N at 2.)

72. This illustrates that AMD's transcoding capabilities are designed and intended to optimize different formats, such as large-screen televisions and small-screen mobile phones. (*See id.* at 2, 5, 16-17.) AMD's dynamic transcoding method therefore satisfies Element 9b.

73. As discussed above, AMD's method is implemented in a manner such that transcoding parameters to be applied to requested media are based on attributes of the end-user client device requesting the material. Thus, for example, when similarly situated client devices request the same content, the transcoding parameters to be applied to the requested media content prior to delivery are the same. Therefore, AMD's dynamic transcoding method satisfies Element 9c.

74. The AMD system transcodes the media content by “perform[ing] additional incremental transcoding on top of the base transcoding.” (Ex. D at 23:66-24:3.) As noted above in 9a, the media collection in AMD has different transcoded versions. (*See supra*, ¶ 70.) These transcoded versions are examples of base transcoded versions. (*See id.*) AMD takes these base versions and “Update[s], Adapt[s], [and] Re-transcode[s]” groups of such videos within a collection. (Ex. N at 13.) These base transcoded versions can be re-transcoded. (*See id.*) Such re-transcoding is additional incremental transcoding on top of the base transcoding. (*See id.*) AMD also enables “live linear streaming” of videos which involves incremental live transcoding of videos upstream of AMD, as Akamai depicts in the graphic below:

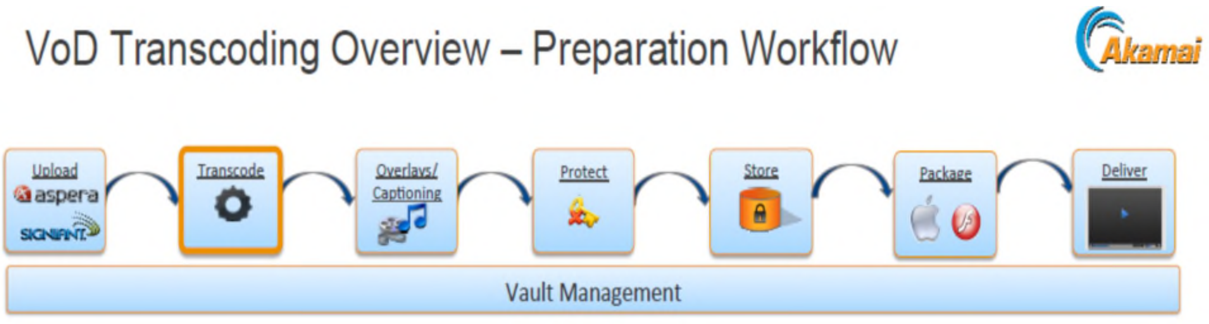

A Transcoding System Designed to Enable 24x7 Live Linear Streaming
Mid Tier (Hot Backup)
Mid Tier (Hot Backup)
Broadcast Site
Encoder
IAS
UDP
Encoder
HTTP(S)
IAT
Ingest Server
Live Transcoding
Mid Tier
Origin Shield
AMD
3rd PARTY
Player
NetStorage
IAS = Ingest Acceleration Source
IAT = Ingest Acceleration Target
~10s for all HTTP(S) Ingested formats

(Ex. O at 1; *see also* Ex. R (developer.akamai.com “Media Delivery” webpage) at 1, Ex. N at 10.) Therefore, AMD’s dynamic transcoding method satisfies Element 9d.

75. In AMD’s dynamic transcoding method, as discussed above, the transcoding includes (1) selecting base transcoded versions of the requested media, which are intermediate media transcoded with only a portion of the transcoding parameters, and (2) creating a final version by incrementally performing further transcoding with the remaining portion of the

transcoding parameters. (*See supra*, ¶¶ 70-74.) As demonstrated above in relation to 9a-d, the Akamai system includes base transcoding and additional incremental transcoding on top of the base transcoding. (*See id.*) The overall transcoding (including the base and additional incremental transcoding on top of the base transcoding) optimizes the media content for different end-user devices such as televisions and mobile phones. (*See* Ex. N at 2.) This optimized media content is the final version of the media content. (*See id.* at 13.) AMD therefore satisfies Elements 9e and its subparts.

76. AMD delivers the transcoded media content to the end-user device in the final "deliver" stage which Akamai depicts as follows:



VoD Transcoding Overview – Preparation Workflow

(*See id.* at 10.) If, for example, two similarly situated client devices request the same transcoded media at substantially the same time, then AMD is designed to deliver the content substantially concurrently. AMD's dynamic transcodiing method therefore satisifes Element 9f.

Front-End Optimization

77. Another service Akamai offers in connection with its Intelligent Edge service is Front End Optimization or "FEO." FEO infringes at least claim 1 of the method claims of the '575 Patent.

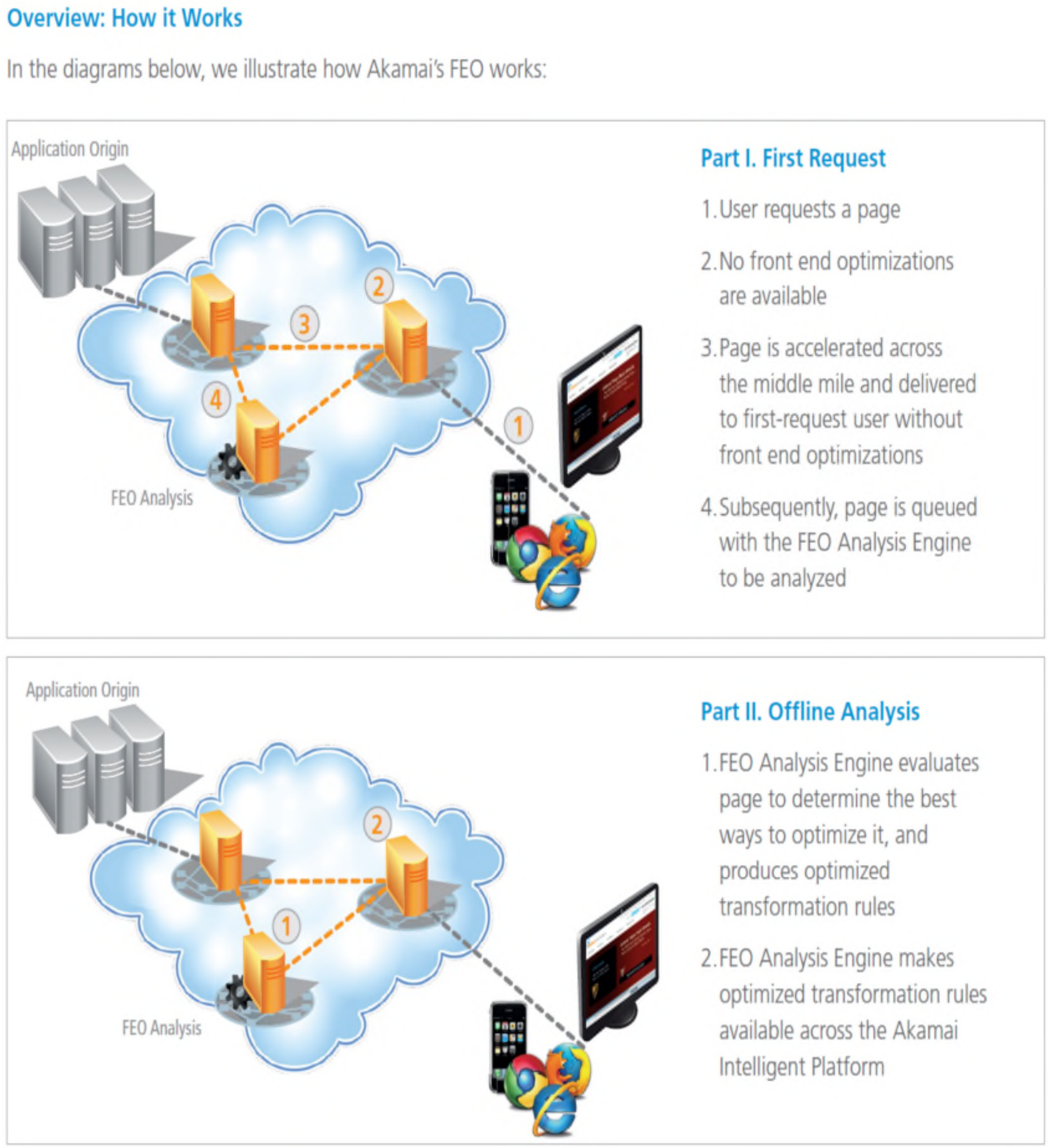
78. Claim 1 of the '575 Patent recites:

[Preamble] A process for delivering an original media, comprising the steps of:

> [1a.] placing said original media in a network system;
>
> [1b.] creating an HTML document or browser language having proprietary media tags and placing said HTML document or browser language onto a Web server;
>
> [1c.] actuating a user request from a customer browser for a Web page by having said Web server pass said requested Web page to an HTML parser on said network system;
>
> [1d.] parsing said HTML document or browser language on said parser by looking for said proprietary media tags;
>
> [1e.] looking up said proprietary media tags in a media tags database on said network system;
>
> [1f.] if at least one media tag of said proprietary media tags is not found,
>
>> [1f(i).] generating a Web media by using said at least one media tag;
>>
>> [1f(ii).] placing said generated Web media in a media cache on said network system;
>>
>> [1f(iii).] converting said at least one media tag to at least one standard HTML equivalent tag that refers to said media in said cache; and
>>
>> [1f(iv).] storing said at least one media tag and said at least one standard HTML equivalent tag in said database;
>
> [1g.] modifying said HTML document or browser language by replacing said at least one media tag by said at least one standard HTML equivalent tag;
>
> [1h.] delivering said modified HTML document or browser language from said network system to said Web server; and
>
> [1i.] delivering said modified HTML document or browser language from said Web server to said customer browser for said customer to view.

79. Akamai’s FEO employs a computer program that performs all of the acts described in claim 1 of the ’575 Patent, and therefore infringes. As discussed in connection with the additional limitations of claim 1, FEO provides a process for delivering an original media. Thus, the ’575 Patent preamble, if limiting, is satisfied.

80. At a high level, Akamai depicts the FEO architecture as follows:



Overview: How it Works

In the diagrams below, we illustrate how Akamai's FEO works:

Part I. First Request

1. User requests a page
2. No front end optimizations are available
3. Page is accelerated across the middle mile and delivered to first-request user without front end optimizations
4. Subsequently, page is queued with the FEO Analysis Engine to be analyzed

Part II. Offline Analysis

1. FEO Analysis Engine evaluates page to determine the best ways to optimize it, and produces optimized transformation rules
2. FEO Analysis Engine makes optimized transformation rules available across the Akamai Intelligent Platform

(Ex. U at 7.) The "FEO Analysis Engine" is designed to analyze webpages in response to a user request and to develop a set of optimization and transformation rules for modifying the web page the next time a user requests it. In the process, the original media, such as images and videos, are placed into the Akamai network system. The transformation rules and website to which they are applied are an HTML document or browser language. FEO employs a media tagging system which Akamai calls "File Versioning." In this system, "[a]ll resources that are optimized are renamed with a unique name. If the object changes, a different uniquely named resource will be created using a proprietary media naming convention." (Ex. T (developer.akamai.com "Optimization Overview" webpage) at 1.) The unique names that Akamai creates using its "proprietary media naming convention" are the same or the equivalent of the claimed "proprietary media tags." Thus, Elements 1a and 1b are satisfied.

81. As shown in the graphic above, once a web page has been analyzed using the FEO Analysis engine, the transformation rules are "made available across the Akamai intelligent platform." This is done in preparation for the system receiving another user request for the analyzed web page. When it does, the request is passed to Akamai's system, which subsequently implements the transformation rules on the web page. The web page and transformation rules are the "HTML document or browser language" of claim 1. Thus, Elements 1c and 1d. are satisfied.

82. Akamai uses its "File Versioning" media tags to ensure that only current media is optimized, delivered, and cached. According to Akamai, media content is stored in a "cache" which its file versioning media tags reference. Akamai uses this system to ensure that old content is not served when new media is available. When the media changes, the old file is no longer referenced. (*See* Ex. U ("Akamai Front-End Optimization on the Akamai Intelligent Platform^{TM}" guide) at 12.) According to Akamai, its FEO Transformer "[m]odifies a copy of the origin's resources based on Analyzer's instructions. Once transformed, these resources are optimized and cached in the edge network. The original files, or base resources, stay untouched and unoptimized on the origin server." (Ex. V ("Akamai Front-End Optimization User Guide for FEO Standard and FEO Premier" user guide) at 7.) This process satisfies Elements 1c - 1f(ii). That is, a user request for a webpage from a browser is actuated when Akamai's FEO parses the webpage data and the transformation rules and looks for Akamai's proprietary file versioning

tags. Using the tags, up-to-date Web media is generated and optimized for the user. The up-to-date media is cached on Akamai's FEO network system.

83. Akamai's FEO Analyzer maintains both a "list of transformations (the actual HTML changes required to optimize the pages)" and a "corresponding list of which resources to transform." The list of resources to transform, which includes media to be optimized, must include data that is the same or equivalent to the "standard HTML equivalent tag" that refers to the media in the cache. Both the list and the Akamai file versioning tags are stored in a database. Thus, Elements 1f(iii) and (iv) are satisfied.

84. When a request for an optimized FEO website is received, Akamai FEO "takes web pages from the origin server, then optimizes and delivers a modified copy of the content to the requesting browser." (*Id.* at 3.) When the modified copy of the content is returned to the requesting browser, the Akamai file versioning tag must be replaced by a standard HTML equivalent tag that the customer's browser can read. Thus, Elements 1g - 1i are satisfied.

85. While Akamai makes some technical information about its services available to the public, as demonstrated above, there is much more information that is not accessible. Discovery may reveal additional Equilibrium Patents that Akamai has infringed, and Equil IP reserves the right to assert such patents in this action.

WILLFUL INFRINGEMENT

86. As set forth above, Akamai was aware of the Equilibrium Patents and was aware of the content, scope, and claims of one or more of the Equilibrium Patents. On information and belief, Akamai knew or was willfully blind to the fact that its activities infringed claims of the Equilibrium Patents.

87. As discussed above, Equilibrium called Akamai's attention to its pending patent applications at the very outset of the parties' discussions in February 2000 in which Equilibrium disclosed to Akamai the Equilibrium technology - *i.e.*, the technology disclosed and claimed in the Equilibrium Patents. At that time, Akamai was a small company. It had less than $4 million in revenue the previous year. Equilibrium met directly with Akamai's lead technical executives. Mr. Segan, who directed his team to take the meeting, stayed with Akamai for 21 years, including serving as CEO. Thus, the individuals who led Akamai's service development at its inception were also aware of Equilibrium virtually at its inception, aware of the capabilities that its service offered, and aware of how those specific capabilities might integrate with and Akamai's service.

88. As also set forth above, Equilibrium was a member of the Akamai "Technology Partner Program" and the Akamai website stated that the Equilibrium MediaRich technology was "state-of-the-art," allowed "companies to reduce production time and expenses," and created "a more engaging customer experience." Thus, Akamai knew that its customers were using Equilibrium's patented service together with Akamai services. Equilibrium Patents are also identified on the Equilibrium website and in Equilibrium service documentation.

89. Furthermore, as also set forth above, one or more of the Equilibrium Patents (including the '575 Patent asserted herein) was cited in prosecution of numerous Akamai patents and, on several occasions, cited by the examiner as a basis for rejecting claims that Akamai sought in prosecution. Specifically, one or more of the Equilibrium Patents were cited in more than twenty Akamai patents and patent applications, from numerous Akamai patent families and directed to numerous aspects of a content delivery system. Akamai's patent prosecution thus indicated that the Equilibrium patents were pervasively relevant to Akamai's activities. On

information and belief, Akamai understood that the Equilibrium Patents were relevant to its activities and, in particular, to its EdgeAdvantage network and successor products. Akamai was also aware from this that Equilibrium's Patents had early priority dates and, in many cases, were material prior art to patent claims that Akamai was trying to get on its competing service.

90. On multiple occasions over the years, Akamai executives had direct discussions with Equilibrium executives about integrating Equilibrium's technology with its own. These discussions specifically referenced Equilibrium's pending and existing patents. Again, Akamai knew about these patents from examiner rejections from its own patent prosecution activities as well as from prior discussions with Equilibrium.

91. Nevertheless, Akamai still decided instead to replicate the features and functionality in competing services without seeking a license to Equilibrium Patents. Moreover, the features, functionalities, and benefits of Akamai's accused services are in relevant respects remarkably like those of Equilibrium's patented services.

92. Given that Akamai was aware of the Equilibrium patents and their relevance to its services, Akamai knew, or ignored an objectively high risk, that replicating features that Equilibrium's service without a license created would risk infringement of the Equilibrium Patents. This was a knowing and flagrant disregard of Equilibrium's patent rights.

THE AMENDED COMPLAINT

93. Equil IP filed this action against Akamai on May 24, 2022. On Akamai's request, the parties agreed to extend Akamai's date to answer or otherwise respond to August 1, 2022. On July 19, 2022, Equil IP asked Akamai to stipulate to the filing of an amended complaint. Equil IP also provided Akamai a draft amended complaint that asserted the same patent claims as are asserted herein. On July 21, 2022, Akamai's counsel said that it was still considering Equil

IP's request. The next day, more than a week before its negotiated deadline, Akamai moved to dismiss. Akamai's motion acknowledged in a footnote Equil IP's intention to amend the complaint.

COUNT I
Infringement of U.S. Patent No. 9,158,745

94. Equil IP repeats and re-alleges each of the allegations in paragraphs 1-93 of this Complaint.

95. Akamai has directly infringed, either literally or under the doctrine of equivalents, at least one method claim of the '745 Patent (including claim 1) at least by using, selling and/or offering for sale Intelligent Edge services incorporating the Image and Video Manager and other features having similar media optimization functionality without the authority of Equil IP.

96. As recited above, Akamai's infringement has been willful, wanton, and deliberate, and in knowing and flagrant disregard of Equil IP's patent rights.

97. Equil IP has been damaged and harmed by Akamai's infringement.

COUNT II
Infringement of U.S. Patent No. 8,495,242

98. Equil IP repeats and re-alleges each of the allegations in paragraphs 1-97 of this Complaint.

99. Akamai has directly infringed, either literally or under the doctrine of equivalents, at least one method claim of the '242 Patent (including claim 9) at least by using, selling and/or offering for sale Intelligent Edge services incorporating the Adaptive Media Delivery and other features having similar transcoding functionality without the authority of Equil IP.

100. As recited above, Akamai's infringement has been willful, wanton, and deliberate, and in knowing and flagrant disregard of Equil IP's patent rights.

101. Equil IP has been damaged and harmed by Akamai's infringement.

COUNT III
Infringement of U.S. Patent No. 6,792,575

102. Equil IP repeats and re-alleges each of the allegations in paragraphs 1-101 of this Complaint.

103. Akamai has directly infringed, either literally or under the doctrine of equivalents, at least one method claim of the ’575 Patent (including claim 1) at least by using, selling and/or offering for sale Intelligent Edge services incorporating Front End Optimization and other features having similar media optimization functionality without the authority of Equil IP.

104. As recited above, Akamai’s infringement has been willful, wanton, and deliberate, and in knowing and flagrant disregard of Equil IP’s patent rights.

105. Equil IP has been damaged and harmed by Akamai’s infringement.

DEMAND FOR JURY TRIAL

Equil IP hereby demands trial by jury on all claims and issues so triable.

PRAYER FOR RELIEF

WHEREFORE, Equil IP respectfully requests that the Court enter judgment in favor of Equil IP and against Akamai as follows:

A. Finding that Akamai has infringed the Asserted Patents.

B. Finding that Akamai’s infringement of the Asserted Patents is willful.

C. Awarding Equil IP the damages it sustained as a result of Akamai’s patent infringement, including, but not limited to, a reasonable royalty.

D. Awarding Equil IP enhanced damages under 35 U.S.C. § 284 as a result of Akamai’s willful patent infringement.

E. Finding this to be an exceptional case under 35 U.S.C. § 285 and awarding Equil IP its attorney’s fees.

F. Awarding Equil IP its costs incurred in this action.

G. Granting Equil IP such other and further relief as the Court may deem just, proper, and appropriate.

Respectfully submitted,

POTTER ANDERSON & CORROON LLP

By: */s/ David E. Moore*
David E. Moore (#3983)
Bindu A. Palapura (#5370)
Brandon R. Harper (#6418)
Carson R. Bartlett (#6750)
Hercules Plaza, 6th Floor
1313 N. Market Street
Wilmington, DE 19801
Tel: (302) 984-6000
dmoore@potteranderson.com
bpalapura@potteranderson.com
bharper@potteranderson.com
cbartlett@potteranderson.com

Attorneys for Plaintiff Equil IP Holdings LLC

OF COUNSEL:

Jason R. Bartlett
Jason A. Crotty
Marc J. Pernick
MAURIEL KAPOUYTIAN WOODS LLP
450 Sansome Street, Suite 1005
San Francisco, CA 94111
Tel: (415) 738-6228

Steven Callahan
Christopher T. Bovenkamp
CHARHON CALLAHAN ROBSON & GARZA, PLLC
3333 Lee Parkway, Suite 460
Dallas, TX 75219
Tel: (214) 521-6400

Dated: August 3, 2022
10284004 / 22225.00001

Exhibit A

US006792575B1

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

(10) **Patent No.: US 6,792,575 B1**
(45) **Date of Patent: Sep. 14, 2004**

(54) AUTOMATED PROCESSING AND DELIVERY OF MEDIA TO WEB SERVERS

(75) Inventors: **Christopher Samaniego**, San Francisco, CA (US); **Nelson H. "Rocky" Offner**, Kensington, CA (US); **Adrian D. Thewlis**, Sausalito, CA (US); **David R. Boyd**, San Francisco, CA (US)

(73) Assignee: **Equilibrium Technologies**, Sausalito, CA (US)

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

(21) Appl. No.: **09/425,326**

(22) Filed: **Oct. 21, 1999**

(51) **Int. Cl.**^{7} **G06F 17/00**; G06F 15/00; G06F 9/30

(52) **U.S. Cl.** **715/513**; 715/501.1; 715/517; 345/735; 709/203; 709/219

(58) **Field of Search** 715/513, 517, 715/501.1; 707/102; 345/735, 800, 629; 709/203, 219

(56) References Cited

U.S. PATENT DOCUMENTS

5,088,052	A	2/1992	Spielman et al.	395/158
5,355,472	A	10/1994	Lewis	395/600
5,530,852	A	6/1996	Meske, Jr. et al.	395/600
5,701,451	A	12/1997	Rogers et al.	395/600
5,708,845	A	1/1998	Wistendahl et al.	395/806
5,710,918	A	1/1998	Lagarde et al.	395/610
5,737,619	A	4/1998	Judson	395/761
5,745,908	A	4/1998	Anderson et al.	707/513
5,793,964	A	8/1998	Rogers et al.	395/200.32
5,822,436	A	10/1998	Rhoads	380/54
5,845,084	A	12/1998	Cordell et al.	395/200.64
5,845,299	A	12/1998	Arora et al.	707/513
5,860,068	A	1/1999	Cook	705/26
5,860,073	A	1/1999	Ferrel et al.	707/522
5,861,881	A	1/1999	Freeman et al.	345/302
5,862,325	A	1/1999	Reed et al.	395/200.31
5,870,552	A *	2/1999	Dozier et al.	709/219
5,880,740	A *	3/1999	Halliday et al.	345/629

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

EP	0747842	A1	12/1996	G06F/17/30
EP	782085		7/1997	G06F/17/30
EP	818907		1/1998	H04L/29/06
EP	0843276	A1	5/1998	G06K/9/20
EP	876034		11/1998	H04L/29/06
EP	833068		12/1998	G06F/17/30
EP	886409		12/1998	H04L/29/06
EP	895171		2/1999	G06F/17/60

OTHER PUBLICATIONS

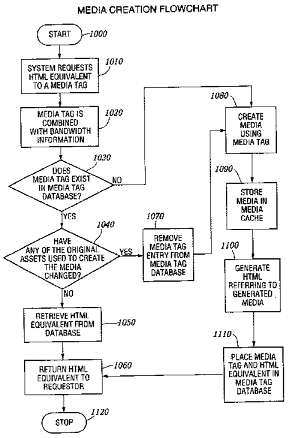
Sakaguchi et al., A browsing tool for multi–lingual documents for users without multi–lingual fonts, ACM International Conference On Digital Libraries, 1996, pp. 63–71.*

Primary Examiner—Sanjiv Shah
Assistant Examiner—William L. Bashore
(74) *Attorney, Agent, or Firm*—Glenn Patent Group; Michael A. Glenn

(57) ABSTRACT

A system using as input original media, an HTML document or browser language having proprietary tags, Web server traffic, and Web-client capabilities to generate an optimized Web media and HTML to refer to the generate media, and to automatically deploy the HTML and media to the Web server is provided. A Web authoring process is provided for facilitating creation of the media, assignment of a unique name to the media, and modification of the HTML document or browser language to contain a proprietary tag. Viewing capability is provided by the Web server passing the HTML or browser language, client browser capabilities, and current server traffic to the system, which parses the HTML or browser language searching for the proprietary tags. If a proprietary tag is found, the tag is processed to generate the Web media. Information is stored in the system database in case identical proprietary tags are processed.

9 Claims, 16 Drawing Sheets


MEDIA CREATION FLOWCHART
START
1000
SYSTEM REQUESTS HTML EQUIVALENT TO A MEDIA TAG
1010
MEDIA TAG IS COMBINED WITH BANDWIDTH INFORMATION
1020
DOES MEDIA TAG EXIST IN MEDIA TAG DATABASE?
1030
NO
YES
HAVE ANY OF THE ORIGINAL ASSETS USED TO CREATE THE MEDIA CHANGED?
1040
YES
NO
REMOVE MEDIA TAG ENTRY FROM MEDIA TAG DATABASE
1070
RETRIEVE HTML EQUIVALENT FROM DATABASE
1050
RETURN HTML EQUIVALENT TO REQUESTOR
1060
STOP
1120
CREATE MEDIA USING MEDIA TAG
1080
STORE MEDIA IN MEDIA CACHE
1090
GENERATE HTML REFERRING TO GENERATED MEDIA
1100
PLACE MEDIA TAG AND HTML EQUIVALENT IN MEDIA TAG DATABASE
1110

U.S. PATENT DOCUMENTS

5,890,170	A	*	3/1999	Sidana	715/501.1
5,895,476	A	*	4/1999	Orr et al.	715/517
5,937,160	A	*	8/1999	Davis et al.	709/203
6,009,436	A	*	12/1999	Motoyama et al.	707/102
6,456,305	B1	*	9/2002	Qureshi et al.	345/800
6,563,517	B1	*	5/2003	Bhagwat et al.	345/735
6,591,280	B2	*	7/2003	Orr	715/513
6,623,529	B1	*	9/2003	Lakritz	715/536

* cited by examiner

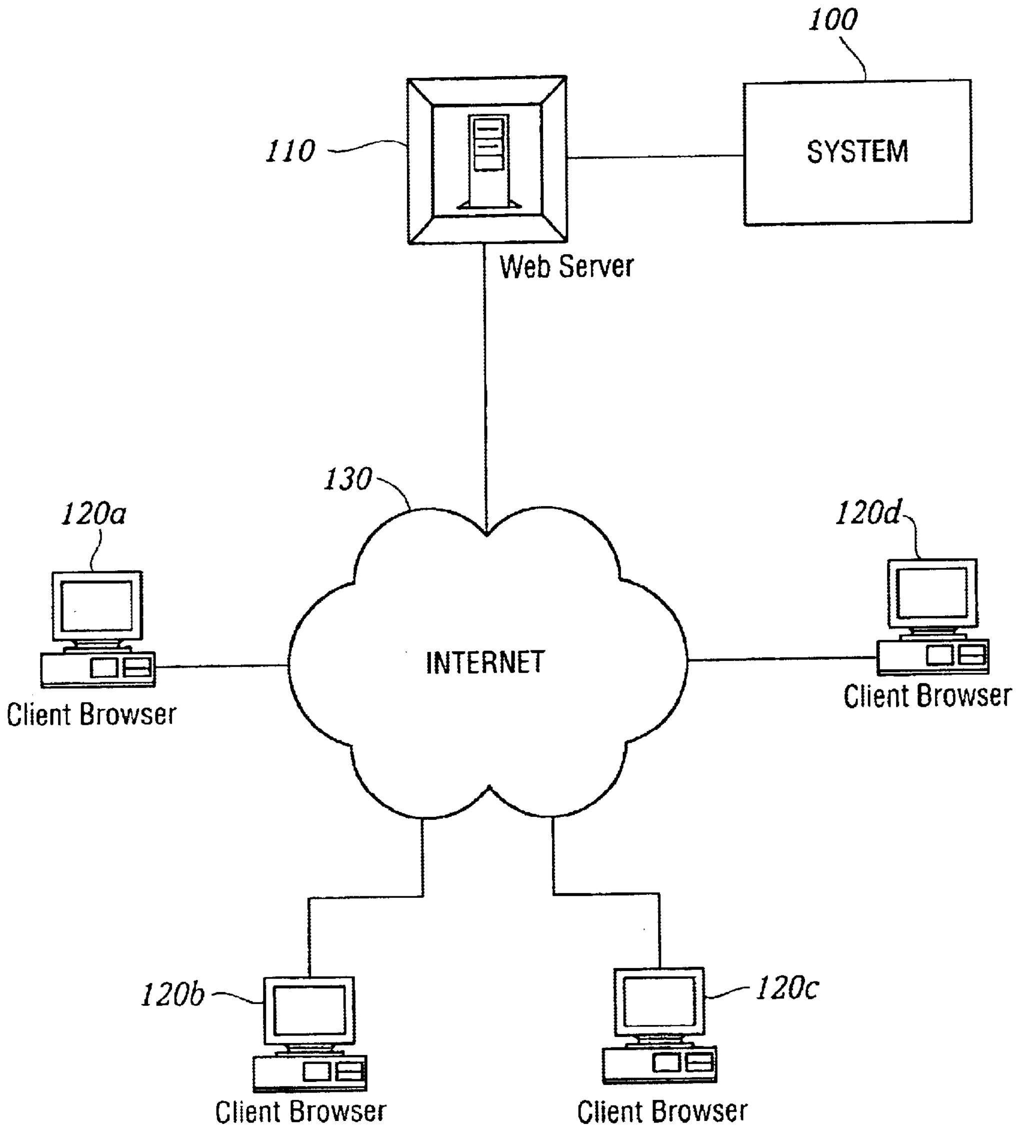

100
110
SYSTEM
Web Server
130
INTERNET
120a
Client Browser
120d
Client Browser
120b
Client Browser
120c
Client Browser

FIG. 1

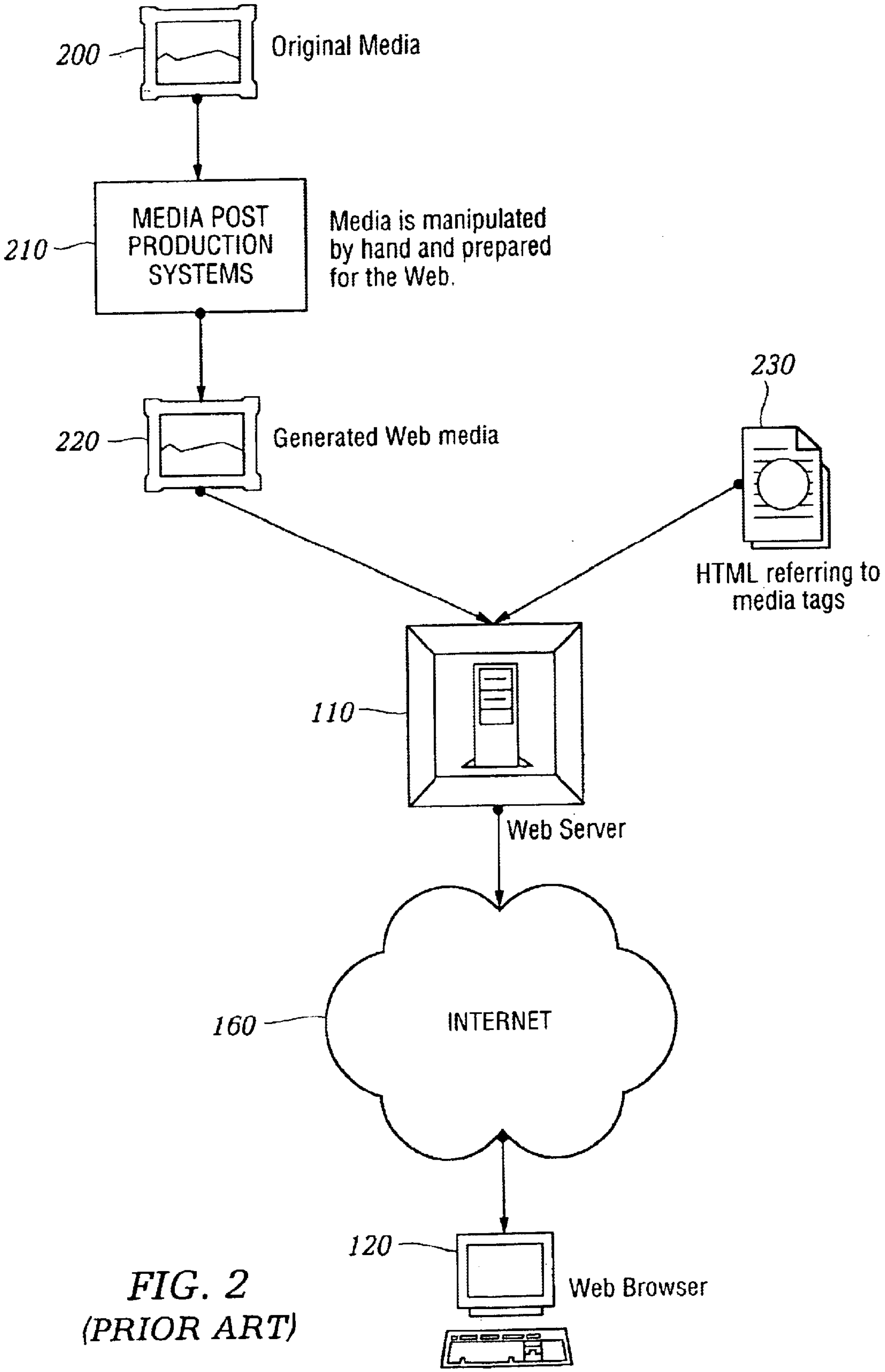

200
Original Media
210
MEDIA POST PRODUCTION SYSTEMS
Media is manipulated by hand and prepared for the Web.
230
220
Generated Web media
HTML referring to media tags
110
Web Server
160
INTERNET
120
Web Browser

FIG. 2
(PRIOR ART)

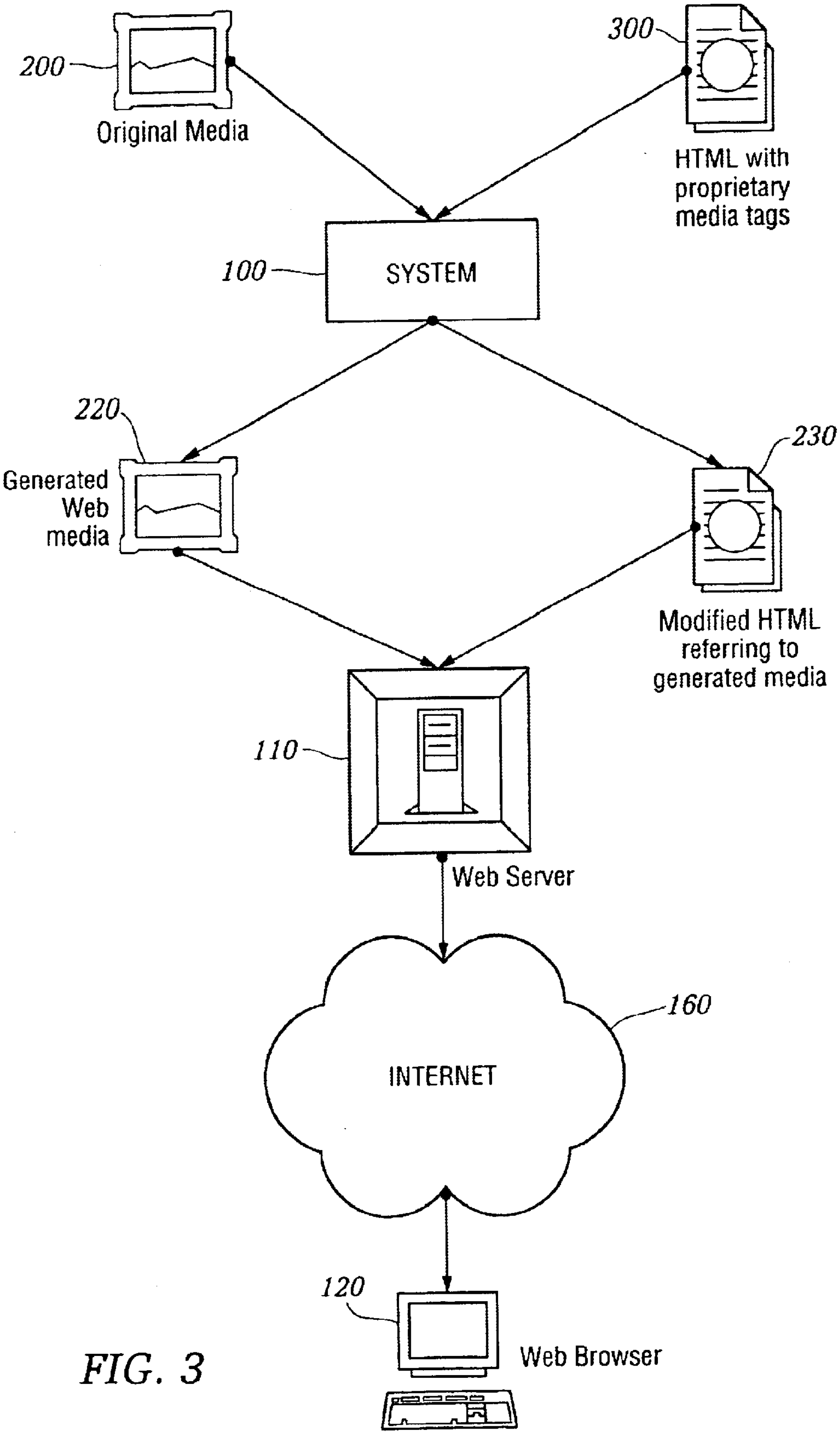

200
Original Media
300
HTML with proprietary media tags
100
SYSTEM
220
Generated Web media
230
Modified HTML referring to generated media
110
Web Server
160
INTERNET
120
Web Browser

FIG. 3

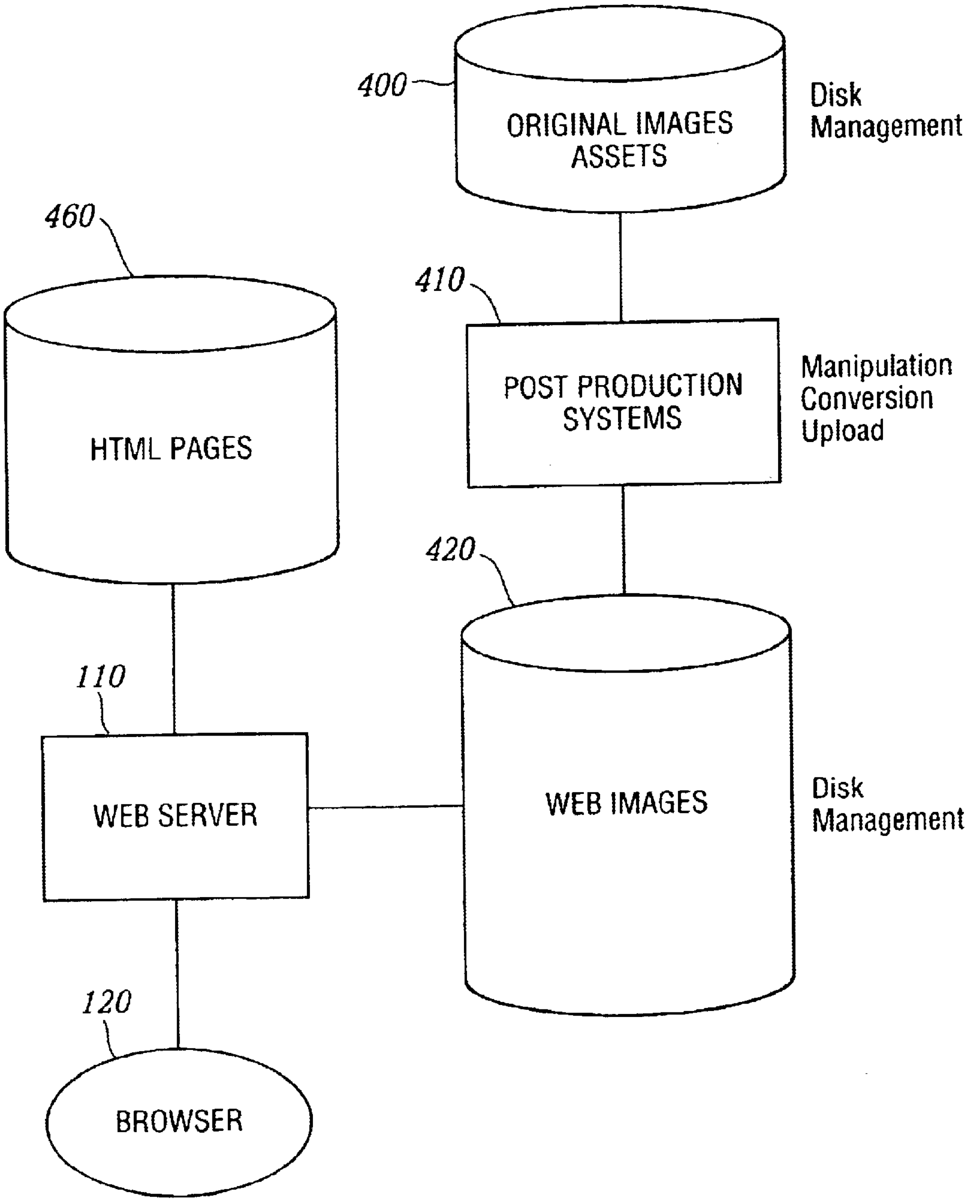


FIG. 4
(PRIOR ART)

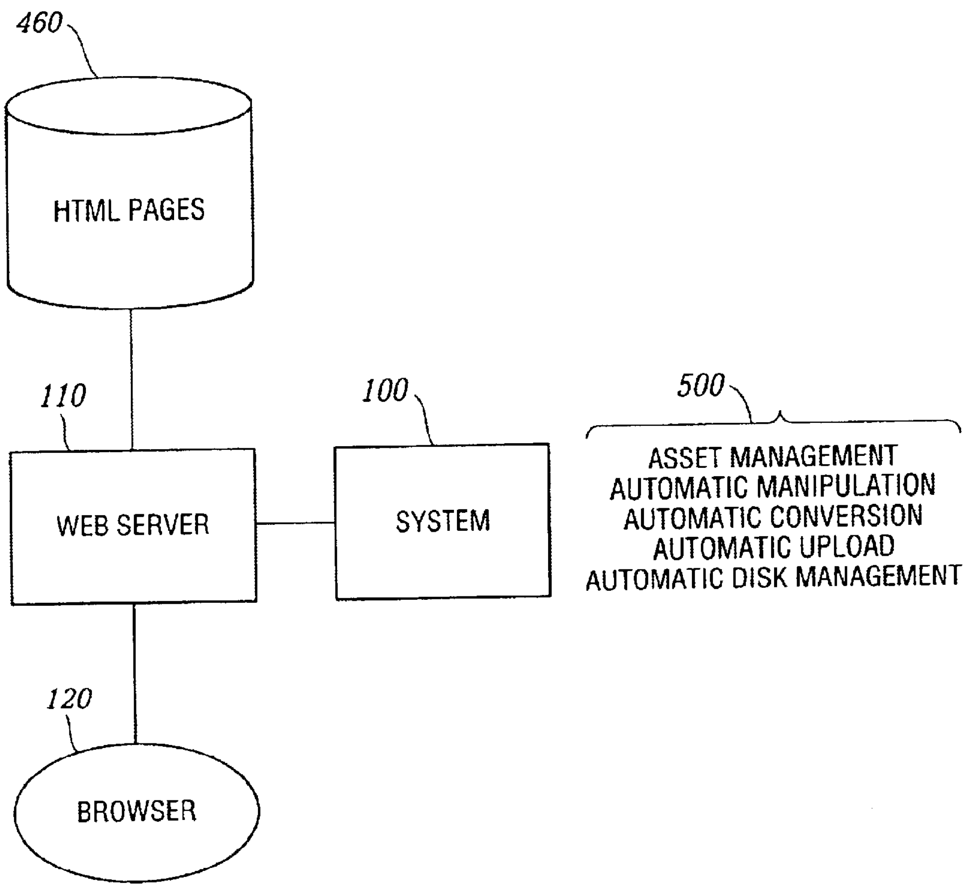

460
HTML PAGES
110
WEB SERVER
100
SYSTEM
500
ASSET MANAGEMENT
AUTOMATIC MANIPULATION
AUTOMATIC CONVERSION
AUTOMATIC UPLOAD
AUTOMATIC DISK MANAGEMENT
120
BROWSER

FIG. 5

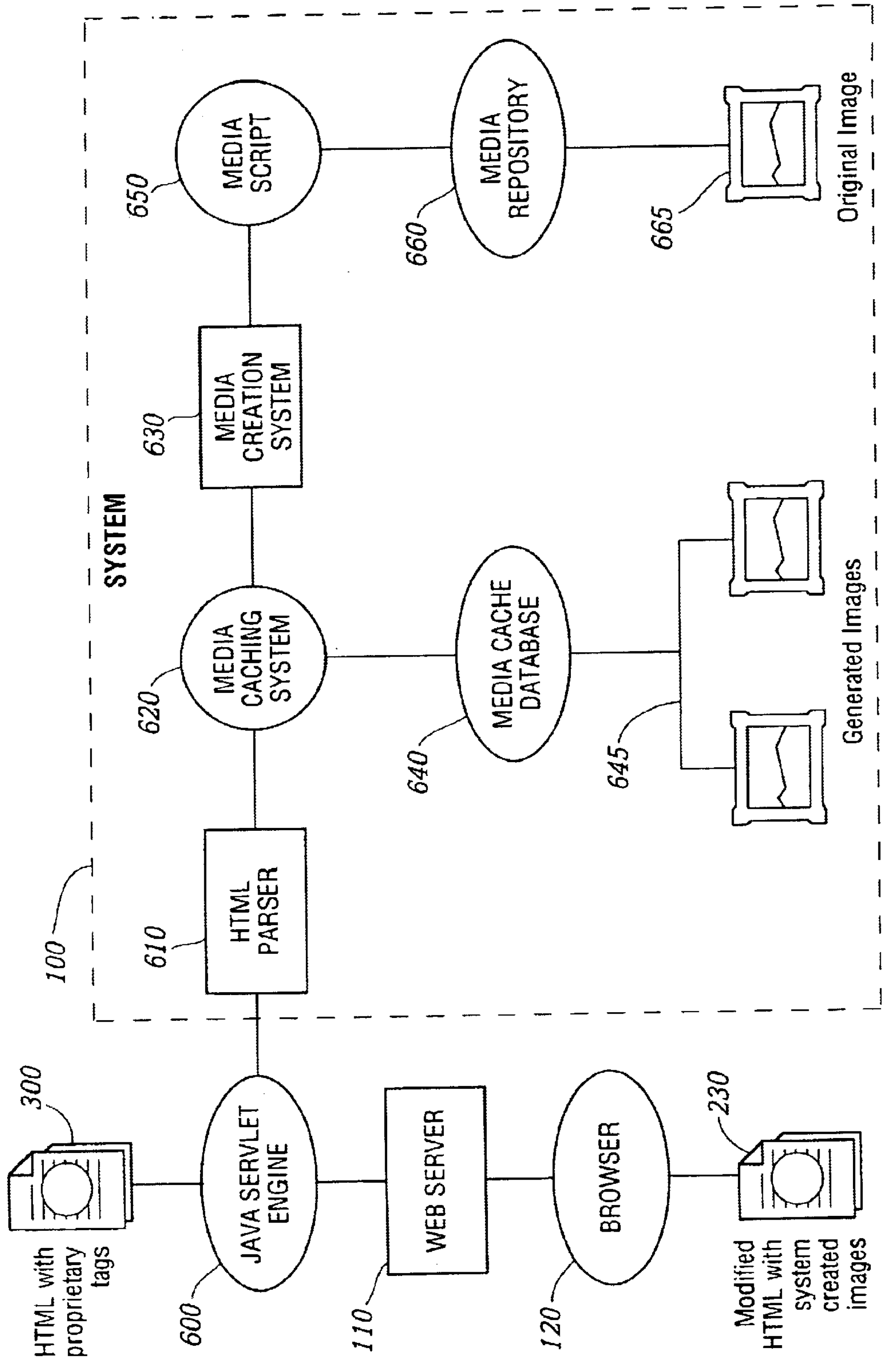

HTML with proprietary tags
300
600
JAVA SERVLET ENGINE
110
WEB SERVER
120
BROWSER
230
Modified HTML with system created images
100
SYSTEM
610
HTML PARSER
620
MEDIA CACHING SYSTEM
630
MEDIA CREATION SYSTEM
650
MEDIA SCRIPT
640
MEDIA CACHE DATABASE
660
MEDIA REPOSITORY
645
Generated Images
665
Original Image

FIG. 6

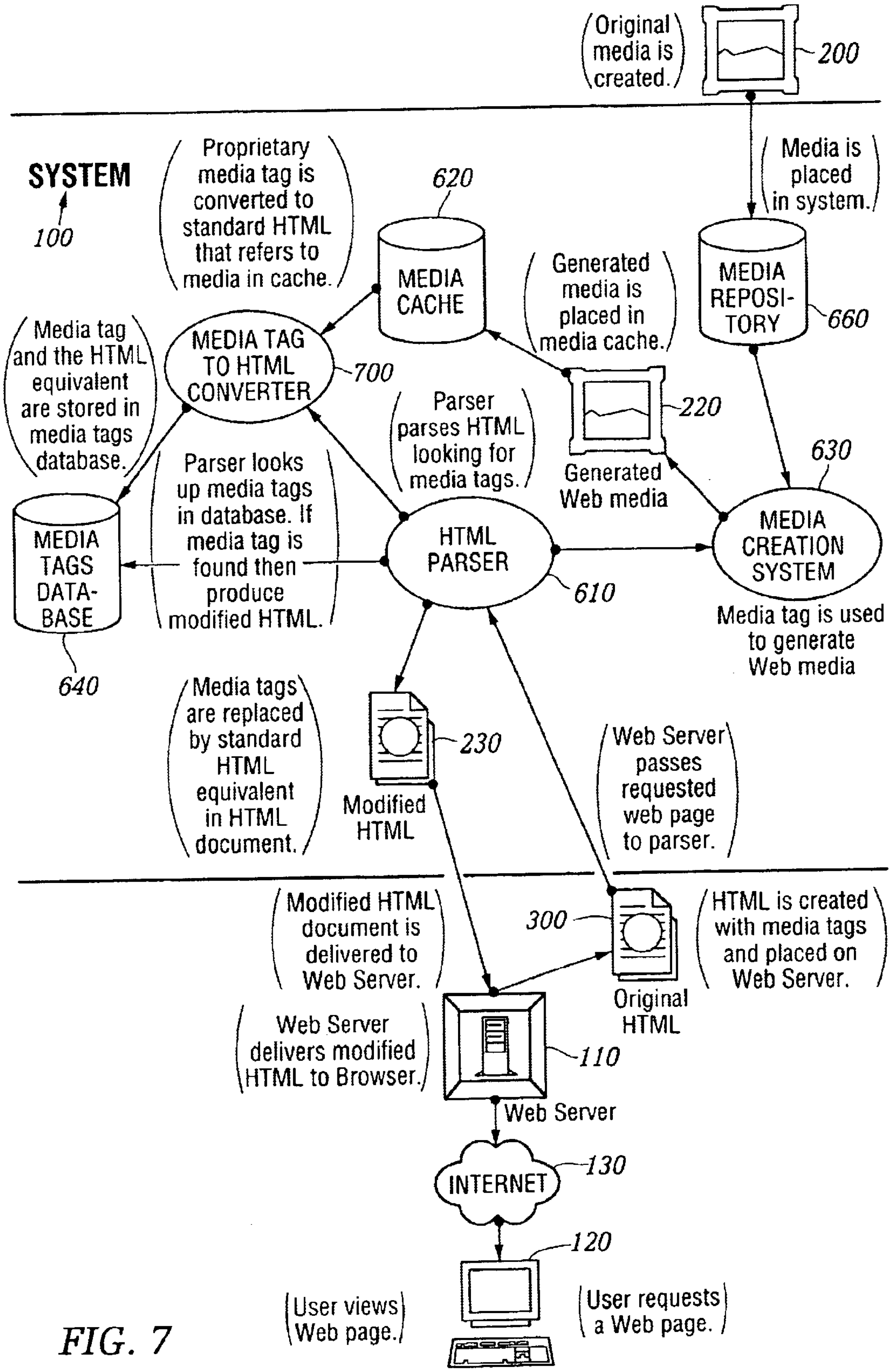

Original media is created.
200
SYSTEM
100
Media is placed in system.
Proprietary media tag is converted to standard HTML that refers to media in cache.
620
MEDIA CACHE
Generated media is placed in media cache.
MEDIA REPOSITORY
660
Media tag and the HTML equivalent are stored in media tags database.
MEDIA TAG TO HTML CONVERTER
700
Parser parses HTML looking for media tags.
220
Generated Web media
630
Parser looks up media tags in database. If media tag is found then produce modified HTML.
MEDIA TAGS DATABASE
HTML PARSER
610
MEDIA CREATION SYSTEM
Media tag is used to generate Web media
640
Media tags are replaced by standard HTML equivalent in HTML document.
230
Modified HTML
Web Server passes requested web page to parser.
Modified HTML document is delivered to Web Server.
300
HTML is created with media tags and placed on Web Server.
Original HTML
Web Server delivers modified HTML to Browser.
110
Web Server
INTERNET
130
120
User views Web page.
User requests a Web page.

FIG. 7

AUTHORING FLOWCHART

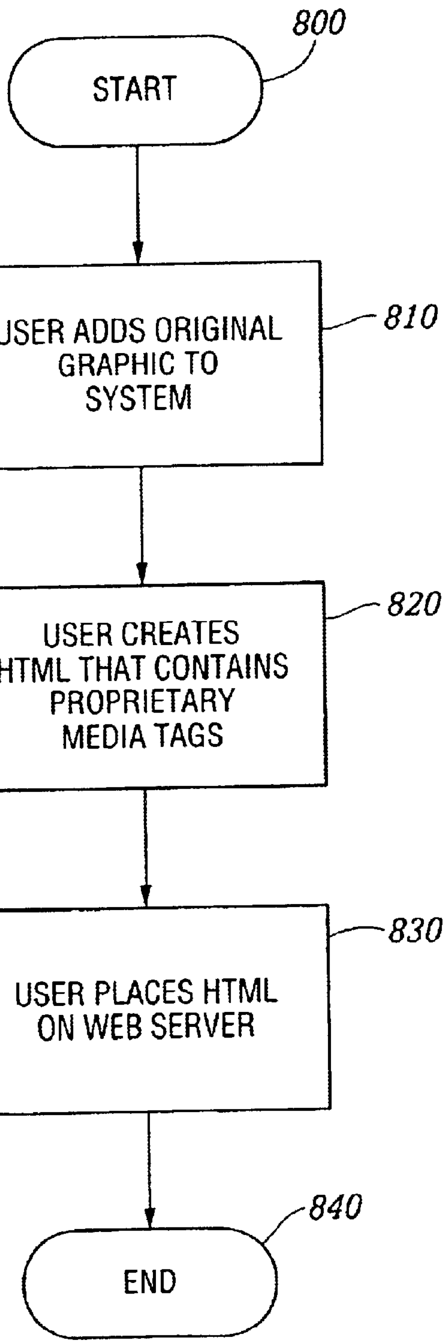

800
START
USER ADDS ORIGINAL GRAPHIC TO SYSTEM
810
USER CREATES HTML THAT CONTAINS PROPRIETARY MEDIA TAGS
820
USER PLACES HTML ON WEB SERVER
830
840
END

FIG. 8

HTML PARSING FLOWCHART

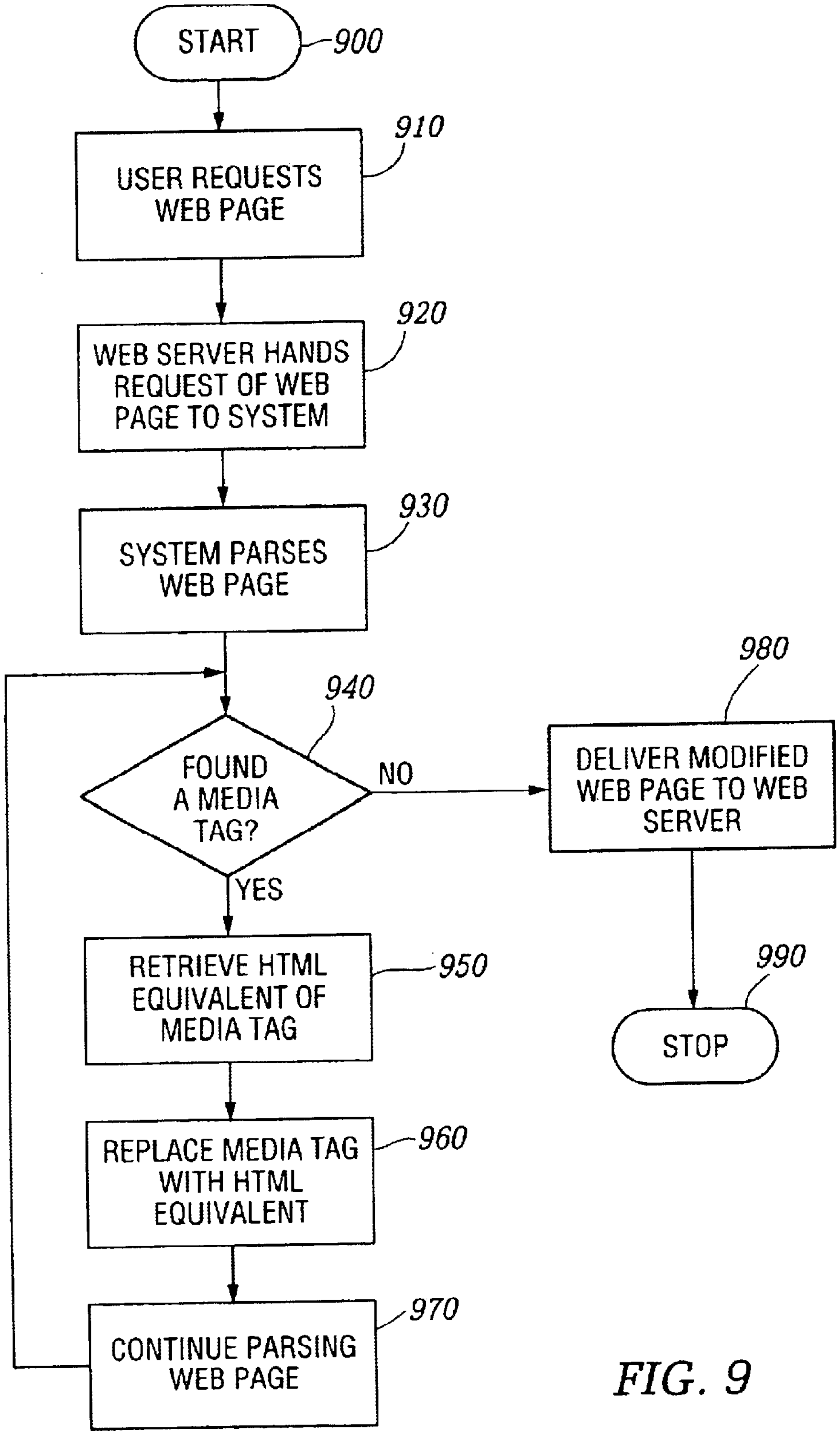

START
900
USER REQUESTS WEB PAGE
910
WEB SERVER HANDS REQUEST OF WEB PAGE TO SYSTEM
920
SYSTEM PARSES WEB PAGE
930
FOUND A MEDIA TAG?
940
NO
YES
DELIVER MODIFIED WEB PAGE TO WEB SERVER
980
STOP
990
RETRIEVE HTML EQUIVALENT OF MEDIA TAG
950
REPLACE MEDIA TAG WITH HTML EQUIVALENT
960
CONTINUE PARSING WEB PAGE
970

FIG. 9

MEDIA CREATION FLOWCHART

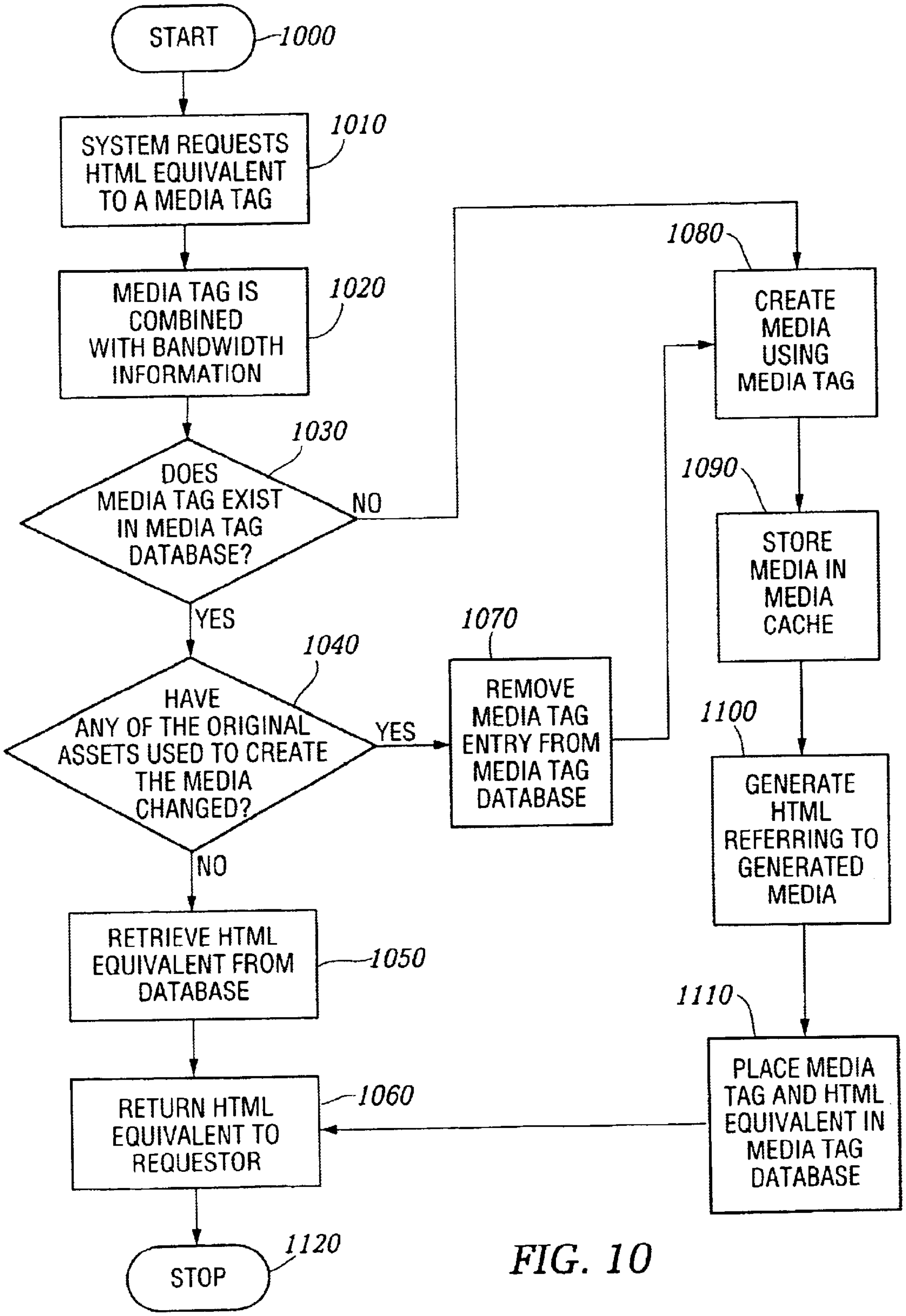
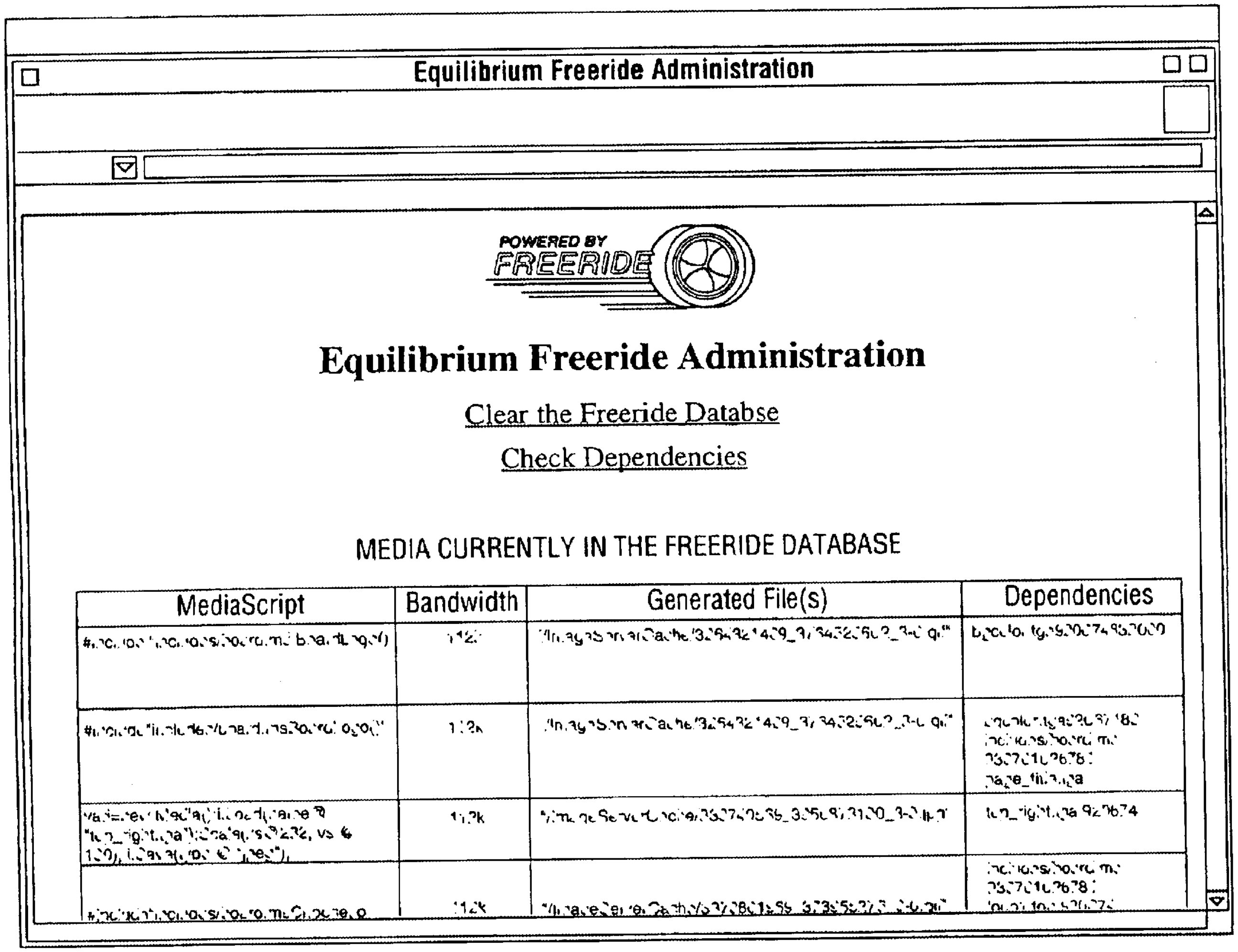

START
1000
SYSTEM REQUESTS HTML EQUIVALENT TO A MEDIA TAG
1010
MEDIA TAG IS COMBINED WITH BANDWIDTH INFORMATION
1020
DOES MEDIA TAG EXIST IN MEDIA TAG DATABASE?
1030
NO
YES
HAVE ANY OF THE ORIGINAL ASSETS USED TO CREATE THE MEDIA CHANGED?
1040
YES
NO
REMOVE MEDIA TAG ENTRY FROM MEDIA TAG DATABASE
1070
CREATE MEDIA USING MEDIA TAG
1080
STORE MEDIA IN MEDIA CACHE
1090
GENERATE HTML REFERRING TO GENERATED MEDIA
1100
PLACE MEDIA TAG AND HTML EQUIVALENT IN MEDIA TAG DATABASE
1110
RETRIEVE HTML EQUIVALENT FROM DATABASE
1050
RETURN HTML EQUIVALENT TO REQUESTOR
1060
STOP
1120

FIG. 10



Equilibrium Freeride Administration

Clear the Freeride Databse

Check Dependencies

MEDIA CURRENTLY IN THE FREERIDE DATABASE

MediaScript	Bandwidth	Generated File(s)	Dependencies
#include "includes/boardname.ms" board_logo()	112k	"/ImageServerCache/3264321409_3734325602_3-0.gif"	board_logo.tga 920674952000
#include "includes/boardname.ms" logo()"	112k	"/ImageServerCache/3264321409_3734325602_3-0.gif"	equalizer.tga 920697182 includes/boardname.ms 932701076781 page_title.tga
var image = new Media(); image.load(name = "top_right.tga"); image.scale(s = 232, vs = 100); image.save(type = "jpeg");	112k	"/ImageServerCache/3527402636_3351973100_3-0.jpg"	top_right.tga 920674
#include "includes/boardname.ms" page_title()	112k	"/ImageServerCache/3272801359_3739552737_3-0.gif"	includes/boardname.ms 932701076781 [illegible]

FIG. 11

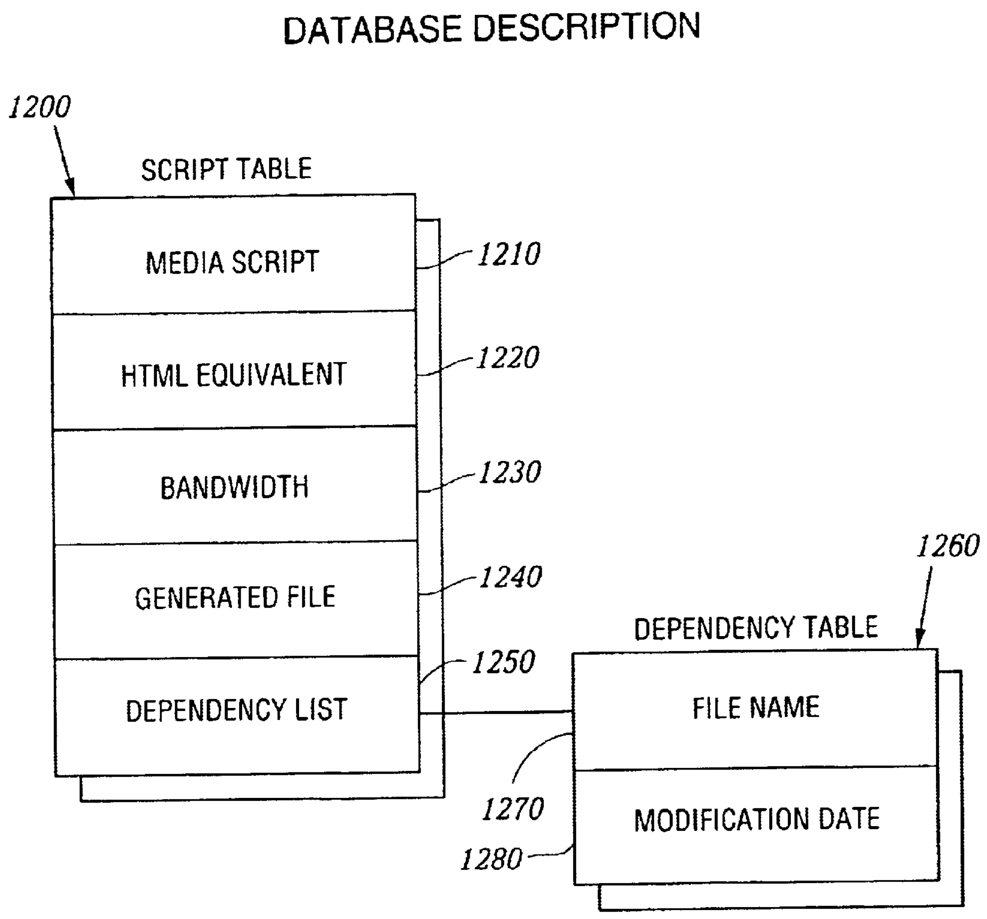

DATABASE DESCRIPTION
1200
SCRIPT TABLE
MEDIA SCRIPT
1210
HTML EQUIVALENT
1220
BANDWIDTH
1230
GENERATED FILE
1240
DEPENDENCY LIST
1250
1260
DEPENDENCY TABLE
FILE NAME
MODIFICATION DATE
1270
1280

FIG. 12

ORIGINAL IMAGES

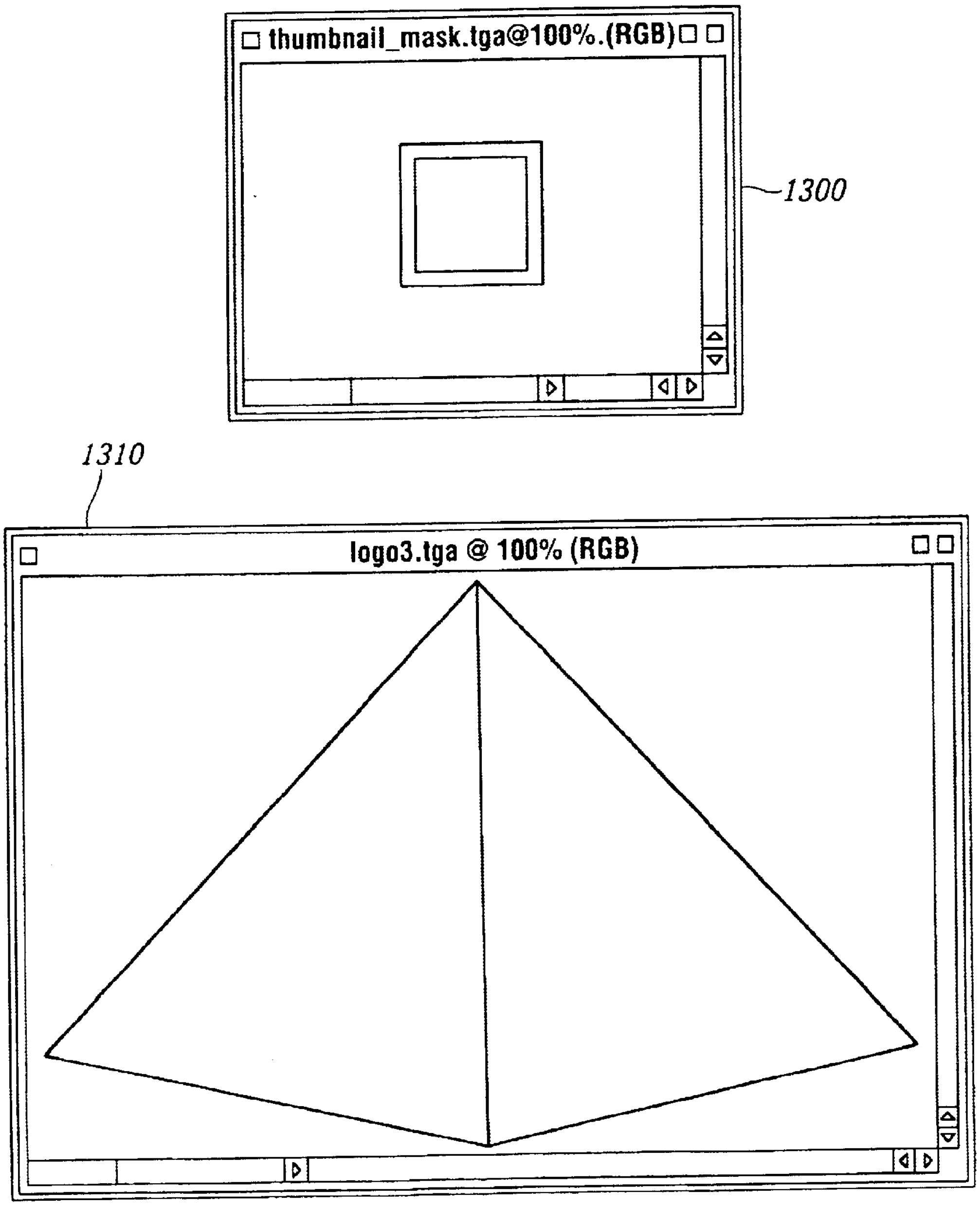
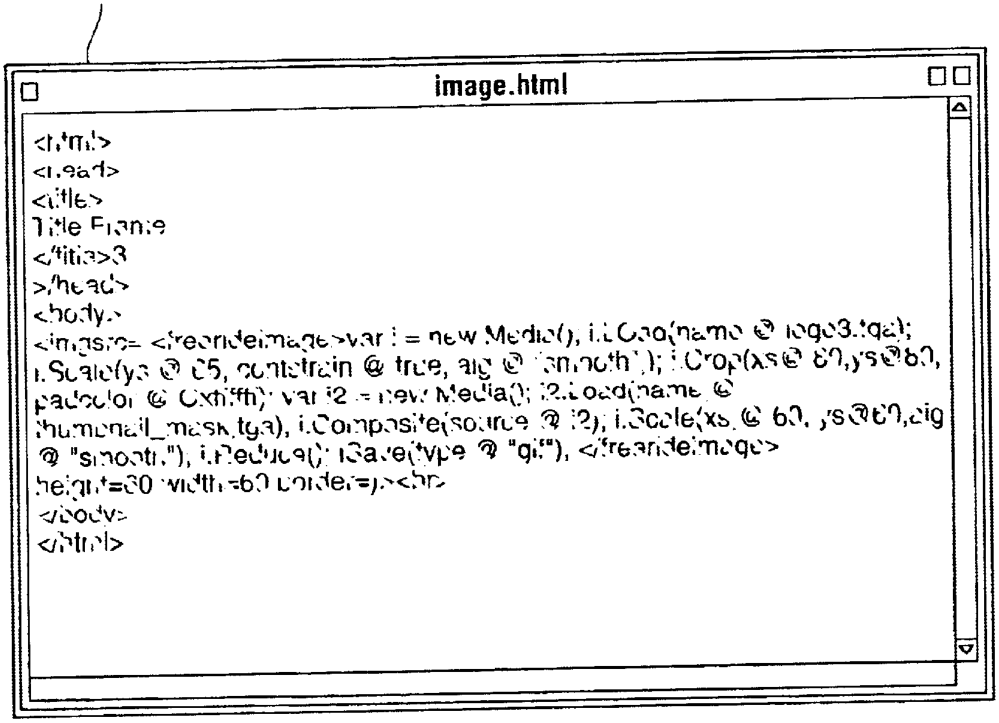


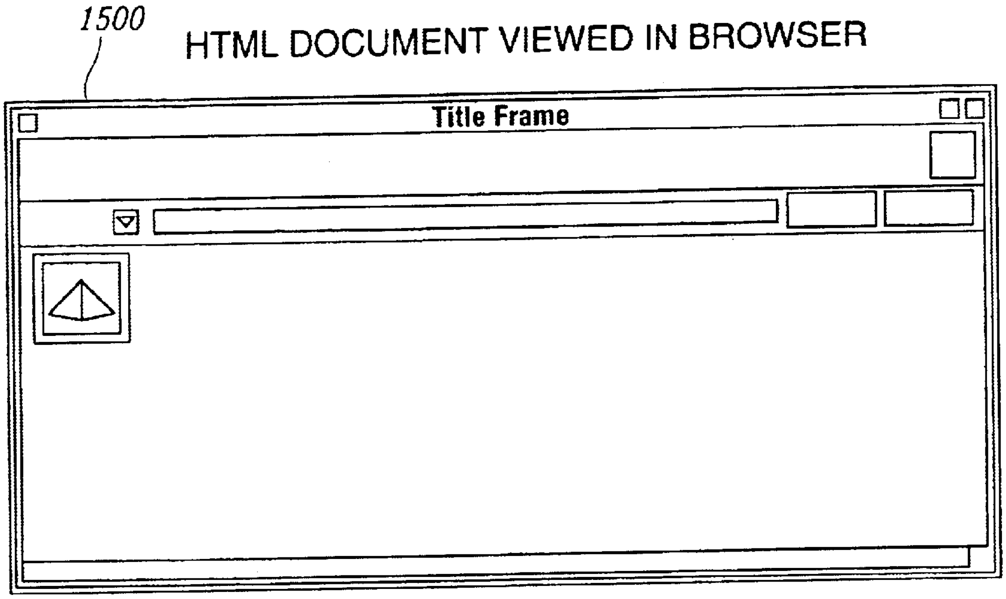
FIG.13

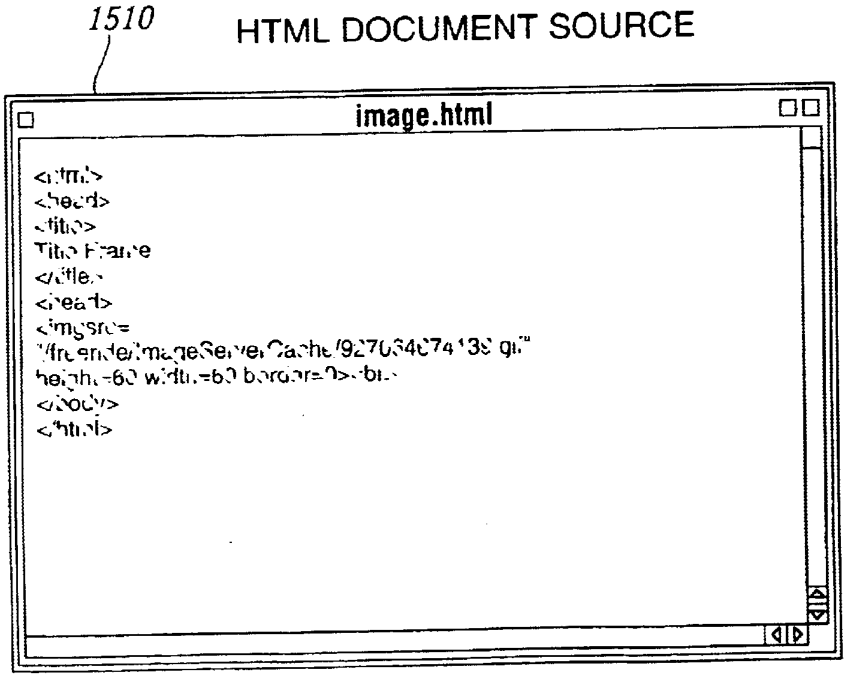
HTML DOCUMENT WITH PROPRIETARY TAG



```
<html>
<head>
<title>
Title Frame
</title>3
>/head>
<body>
<imgsrc= <freerideimage>var i = new Media(); i.Load(name @ logo3.tga);
i.Scale(ys @ 65, constrain @ true, alg @ "smooth"); i.Crop(xs@ 60,ys@60,
padcolor @ 0xffffff); var i2 = new Media(); i2.Load(name @
thumbnail_mask.tga), i.Composite(source @ i2); i.Scale(xs @ 60, ys@60,alg
@ "smooth"); i.Reduce(); i.Save(type @ "gif"), </freerideimage>
height=60 width=60 border=)><br>
</body>
</html>
```

FIG. 14





```
<html>
<head>
<title>
Title Frame
</title>
</head>
<imgsrc=
"/frontend/ImageServerCache/927034074139.gif"
height=60 width=60 border=0><br>
</body>
</html>
```

FIG.15

GENERATED GIF IMAGE

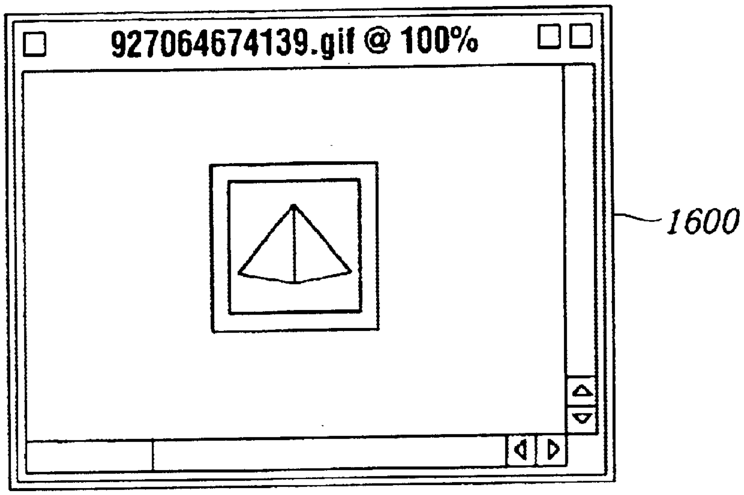


FIG.16

AUTOMATED PROCESSING AND DELIVERY OF MEDIA TO WEB SERVERS

BACKGROUND OF THE INVENTION

1. Technical Field

The invention relates to software systems. More particularly, the invention relates to an Internet server-based software system that provides delivery of automated graphics and other media to Web sites for access by an end user or consumer.

2. Description of the Prior Art

Most Web sites today are primarily handmade. From the guy publishing a simple online technology newsletter from his home, to the Fortune 1000 company's multi-tiered site with hundreds of pages of text, images, and animations, the Web developer and each of his HTML-coding and graphics-producing coworkers toil page by page and image by image. Thousands of established online companies employ hundreds of highly-skilled workers just to produce and maintain their Web sites. After all, the Web is now a major selling vehicle and marketing medium for many of these companies. The Web has even sprouted service industries such as, for example, public companies with multi-billion dollar valuations created just to consult and produce Web sites for others.

Most Web developers who use established WYSIWYG tools in the industry still must produce each page on their Web site one by one. The same rate applies to preparing and placing images, animations, and other visual assets. Each page represents its own set of issues ranging from whether to use GIF, JPEG, or PNG file formats, to finding the optimum bit depth for each image to ensure the fastest downloading through the different browsers of the consumer. The bottlenecked state of the customer's workflow to produce graphics for Web pages can be described as follows:

Current Workflow for Creating Web Graphics

Original Artwork/Asset Creation
  Use third-party point products

Asset Editing
  Scale/reduce/slice

Asset Format Conversion
  JPEG/GIF/PNG

Asset Staging
  Place in Web file system
  Edit HTML

Create/Modify HTML for particular page

Store HTML on Web server

View final pages

Repeat process for each version of each graphic on each page

Estimated time
  Two hours per page times the number of pages

Also, from a user's perspective, the current state of the art is to offer the consumer zooming and panning capabilities so that by clicking on an image the consumer can view more closely or from a different angle. On the horizon are pages with three-dimensional imagery that enable a user to move around a page that can look more like a room than a brochure. While interesting, these features are merely incremental improvements to a consumer's surfing experience.

D. C. A. Bulterman, *Models, Media, and Motion: Using the Web to Support Multimedia Documents*, Proceedings of

1997 International Conference on Multimedia Modeling, Singapore, 17–20 Nov. 1997 discloses "an effort underway by members of industry, research centers and user groups to define a standard document format that can be used in conjunction with time-based transport protocols over the Internet and intranets to support rich multimedia presentations. The paper outlines the goals of the W3C's Synchronized Multimedia working group and presents an initial description of the first version of the proposed multimedia document model and format."

Text and Graphics on UMI's ProQuest Direct. The Best (*yet*) *of both Worlds*, Online, vol. 21, no. 2, pp. 73–7, March–April 1997 discloses an information system that offers "periodical and newspaper content covering a wide range of business, news, and professional topics . . . letting the user search both text and graphics and build the product to suit. Articles can be retrieved in varying levels of detail: citation, abstracts, full text, and text with graphics. Images come in two flavors: Page Image, a virtual photocopy, and Text+Graphics, in which graphics are stored separately from the text and are manipulable as discrete items. . . . [The system] comes in two versions: Windows and Web."

John Mills Dudley, Network-Based Classified Information Systems, AU-A-53031/98 (Aug. 27, 1998) discloses a "system for automatically creating databases containing industry, service, product and, subject classification data, contact data, geographic location data (CCG-data) and links-to web pages from HTML, XML, or SGML encoded web pages posted on computer networks such as Internets or Intranets. . . . The . . . databases may be searched for references (URLs) to web pages by use of enquiries which reference one or more of the items of the CCG-data. Alternatively, enquiries referencing the CCG-data in the databases may supply contact data without web page references. Data duplication and coordination is reduced by including in the web page CCG-data display controls which are used by web browsers to format for display the same data that is used to automatically update the databases."

Cordell et al, Automatic Data Display Formatting with A Networking Application, U.S. Pat. No. 5,845,084 (Dec. 1, 1998) discloses a placeholder image mechanism. "When a data request is made, the data transfer rate is monitored. When the receive data transfer rate is slow, and the data contains an embedded graphical image of unknown dimensions, a small placeholder image is automatically displayed for the user instead of the actual data. The small placeholder image holds a place on a display device for the data or the embedded graphical image until the data or embedded graphical image is received. When embedded graphical image is received, the placeholder image is removed, and the display device is reformatted to display the embedded graphical image."

Jonathon R. T. Lewis, System For Substituting Tags For Non-Editable Data Sets In Hypertext Documents And Updating Web Files Containing Links Between Data Sets Corresponding To Changes Made To The Tags, U.S. Pat. No. 5,355,472 (Oct. 11, 1994) discloses a "hypertext data processing system wherein data sets participating in the hypertext document may be edited, the data processing system inserting tags into the data sets at locations corresponding to the hypertext links to create a file which is editable by an editor and the data processing system removing the tags, generating a revised data set and updating the link information after the editing process. Its main purpose is to preserve the linking hierarchy that may get lost when the individual data sets get modified."

Wistendahl et al, System for Mapping Hot Spots in Media Content Interactive Digital Media Program, U.S. Pat. No.

5,708,845 (Jan. 13, 1998) discloses a "system for allowing media content to be used in an interactive digital media (IDM) program [that] has Frame Data for the media content and object mapping data (N Data) representing the frame addresses and display location coordinates for objects appearing in the media content. The N Data are maintained separately from the Frame Data for the media content, so that the media content can be kept intact without embedded codes and can be played back on any system. The IDM program has established linkages connecting the objects mapped b y the N Data to other functions to be performed in conjunction with display of the media content. Selection of an object appearing in the media content with a pointer results in initiation of the interactive function. A broad base of existing non-interactive media content, such as movies, videos, advertising, and television programming can be converted to interactive digital media use. An authoring system for creating IDM programs has an object outlining tool and an object motion tracking tool for facilitating the generation of N Data. In a data storage disk, the Frame Data and the N Data are stored on separate sectors. In a network system, the object mapping data and IDM program are downloaded to a subscriber terminal and used in conjunction with presentation of the media content."

Rogers et al, Method for Fulfilling Requests of A Web Browser, U.S. Pat. No. 5,701,451 (Dec. 23, 1997) and Lagarde et al, Method for Distributed Task Fulfillment of Web Browser Requests, U.S. Pat. No. 5,710,918 (Jan. 20, 1998) disclose essentially "improvements which achieve a means for accepting Web client requests for information, obtaining data from one or more databases which may be located on multiple platforms at different physical locations on an Internet or on the Internet, processing that data into meaningful information, and presenting that information to the Web client in a text or graphics display at a location specified by the request."

Tyan et al, HTML Generator, European Patent Application No. EP 0843276 (May 20, 1998) discloses "generating an HTML file based on an input bitmap image, and is particularly directed to automatic generation of an HTML file, based on a scanned-in document image, with the HTML file in turn being used to generate a Web page that accurately reproduces the layout of the original input bitmap image."

TrueSpectra has a patent pending for the technology employed in its two products, IrisAccelerate and IrisTransactive. These products are designed for zooming and panning and simple image transformations and conversions, respectively. They support **10** file formats and allow developers to add new file formats via their SDK. They do not require the use of Flashpix for images. However, their documentation points out that performance is dependent on the Flashpix format. The system would be very slow if a non-Flashpix format was used.

TrueSpectra allows the image quality and compression to be set for JPEGs only. The compression setting is set on the server and all images are delivered at the same setting.

TrueSpectra has a simple caching mechanism. Images in the cache can be cleared out automatically at certain times and it does not have any dependency features for image propagation. The Web server needs to be brought down in order to update any original assets.

TrueSpectra does not require plug-ins to operate features such as zooming/panning or compositing, but suggests using their plug-ins for better performance. The alternative to plug-ins is using their Javascript or active server page technology. These technologies are used by many Web sites to provide interactivity, but not all Web browsers work correctly with these technologies.

In their latest version, TrueSpectra fixed their software so that communication to the server will not have to be through a certain port. Using TrueSpectra without a port requires the system to use Javascript or Active Server Pages. This requirement is not always possible so firewall issues remain.

TrueSpectra relies on Flashpix as its native file format and does not support media types such as multi-GIFs and sound formats. Flashpix files are typically larger than most file formats. Access to files is faster for zooming and panning, but appears to be quite slow.

The key to IrisTransactive is the compositing subsystem. It requires three things to build a shopping solution using image composition.

1) The original images must be created. It is suggested that the image be converted to Flashpix for better performance.
2) All of the individual images must be described in XML using the image composer program. The program allows the editor to specify anchor points, layer attributes, and layer names. The resulting file is between 5 k and 50 k.
3) The Web designer must place HTML referring to the XML in the Web site. By specifying parameters to the XML, the Web designer can turn on or off layers.

The herein above process for compositing images enables Web designers to create shopping sites. However, a lot of overhead is the result. The XML documents add 5 k–50 k to a Web site. The compositing commands that are embedded in the HTML are difficult to understand. And, because the compositing feature requires several steps to implement, it is not suitable for every image on a Web site. The process seems to be designed for the specific purpose of shopping.

The disclosed prior art fail to provide systems and mhethodologies that result in a quantum leap in the speed with which they can modify and add images, video, and sound to sites, in the volume of data they can publish internally and externally, and in the quality of the output. The development of such an automated media delivery system would constitute a major technological advance.

It would be advantageous to empower an end user with flexibility and control by providing interactive page capabilities.

It would be advantageous from an end user's perspective to generate Web pages that contain active graphics. For example, clicking on a Corvette image will cause a simple menu to pop up suggesting alternative colors and sizes in which to see the car. Clicking on portions of the image, such as a fender, can call up a close-in view of the fender.

It would be advantageous to provide an automated graphics delivery system that becomes part of the Web site infrastructure and operates as part of the Web page transaction and that thereby provides a less expensive and less time-consuming process.

It would be advantageous to provide a system for automated processing and delivery of media (images, video, and sound) to a Web server whereby it eliminates the laborious post-production and conversion work that must be done before a media asset can be delivered on a Web server.

It would be advantageous to create a dynamic Web site, wherein images are generated on demand from original assets, wherein only the original assets need to be updated, and wherein updated changes propagate throughout the site.

It would be advantageous to provide a system that generates media based on current Web server traffic thereby optimizing throughput of the media through the Web server.

It would be advantageous to provide a system that generates media that is optimized for the Web client, wherein client connection speed determines optimum quality and file size.

It would be advantageous to provide a system that generates media, whereby the media is automatically uploaded.

It would be advantageous to provide a system that automatically caches generated media so identical requests can be handled without regeneration of images.

It would be advantageous to provide a system that resides behind the Web server, thereby eliminating security issues.

It would be advantageous to provide a system wherein the client browser does not require a plug-in.

It would be advantageous to provide a system wherein the system does not require any changes to a Web server.

It would be advantageous to provide a system wherein the system manages the Web server media cache.

It would be advantageous to provide a system wherein the Web media is generated only if requested by a client browser.

SUMMARY OF THE INVENTION

An automated media delivery system that becomes part of the Web site infrastructure and operates as part of the Web page transaction is provided. The claimed invention streamlines the post-production process by automating the production of a media through proprietary HTML tags embedded in Web documents. The author simply places the original media in the system and adds proprietary HTML tags to HTML or other browser language. The system automatically processes the proprietary HTML tags and produces the media for the Web client. It also replaces the proprietary HTML tags with standard HTML tags so to be processed correctly by the HTML client.

This invention takes as input the client connection, server traffic, and proprietary HTML tags in order to generate the optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the need is automatically handled by the claimed invention. The generated media is cached so that further requests for the same media require little overhead.

Because the invention takes the original media and proprietary HTML tags as inputs for generating the Web media, it is possible to modify one or both and have the system automatically update the media on all of the associated Web pages.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a schematic diagram showing the placement of the system within a current Web infrastructure according to the invention;

FIG. **2** is a schematic diagram showing how a typical Web site delivers an HTML document and its graphics to a Web browser according to the prior art;

FIG. **3** is a schematic diagram showing delivery of an HTML document and media to a Web browser according to the invention;

FIG. **4** is a schematic diagram showing the components involved in Web site administration according to the prior art;

FIG. **5** is a schematic diagram showing the components of the system involved in Web site administration according to the invention;

FIG. **6** is a simple overview showing the components of the system according to the invention;

FIG. **7** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to the invention;

FIG. **8** is a flow chart showing an authoring process according to the invention;

FIG. **9** is a flow chart showing an HTML parsing process according to the invention;

FIG. **10** is a flow chart showing a media creation process according to the invention;

FIG. **11** is a screen shot showing an administration tool according to the invention;

FIG. **12** displays a structure of a database record used for the system according to the invention;

FIG. **13** shows original media to be processed according to the invention;

FIG. **14** shows a portion on an HTML document with a proprietary tag according to the invention;

FIG. **15** shows an HTML document and an HTML document source according to the invention; and

FIG. **16** shows a generated GIF image according to the invention.

DETAILED DESCRIPTION OF THE INVENTION

An automated graphics delivery system that becomes part of the Web site infrastructure and operates as part of the Web page transaction is provided. The claimed invention streamlines the post-production process by automating the production of a media through proprietary HTML tags embedded in Web documents. The author simply places the original media in the system and adds proprietary HTML tags to HTML documents. The system automatically processes the proprietary HTML tags and produces the media for the Web client. It also replaces the proprietary HTML tags with standard HTML tags so it can be processed correctly by the HTML client.

This invention takes as input the client connection, server traffic, and proprietary HTML tags in order to generate the optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the need is automatically handled by the claimed invention. The generated media is cached so that further requests for the same media require little overhead.

FIG. **1** is a schematic diagram showing the placement of the system within a current Web infrastructure according to a preferred embodiment of the invention. The system **100** is attached to a Web server **110**, which is connected to multiple client browsers **120**(*a–d*) via the Internet **130**.

FIG. **2** is a schematic diagram showing how a typical Web site delivers an HTML document and its graphics to a Web browser according to the prior art. An original media **200** is passed to post-production systems **210**, wherein the media **200** is manipulated by hand and prepared for the Web. The result is a Web media **220**. The Web media **220** and an associated HTML document **230** referring to the media **220** by media tags are input to a Web server **110** for a Web browser **120** to view via the Internet **130**.

FIG. **3** is a schematic diagram showing delivery of an HTML document and media to a Web browser according to a preferred embodiment of the invention. An original media **200** and an HTML document embedded with proprietary media tags **300** are input into the system **100**. The system **100** generates a Web-safe media **220** and a modified HTML document **230** that refers to the Web media, and automatically loads them onto the Web server **110** for view by a Web browser **120** via the Internet **160**.

FIG. **4** is a schematic diagram showing components involved in Web site administration according to the prior art. Original media assets **400** are original images, video, or sound that have not been prepared for the Web. Web sites usually need to manage the placement of media on the network for easy retrieval by Web designers. Post-production systems **410** vary from Web site to Web site. Post-production systems **410** are usually custom procedures that Web designers use to convert an original media, such as an image, to one that can be displayed on the Web. Post-production systems **410** also upload finished images to Web image systems. Web images **420** are Web versions of the original images. Web images **420** are ready for retrieval by the Web server **110** to be delivered to a Web browser **120**. Any image to be modified or updated must pass through the herein above three components before it can be delivered to the Web browser **120**. HTML pages **460** have references to Web images **420**.

FIG. **5** is a schematic diagram showing the components involved in Web site administration according to a preferred embodiment of the invention. Web site administration is simplified using the claimed invention. Asset management, automatic image manipulation, automatic image conversion, automatic image upload, and automatic disk management **500** are provided by the claimed invention.

FIG. **6** is a simple overview showing the components of the system according to a preferred embodiment of the invention. HTML with proprietary tags **300** is the original HTML document that is embedded with proprietary tags which describe how the images are to be manipulated for the Web. Java servlet engine **600** is a third-party product that allows the system **100** to interface with the Web server **110** and execute Java servlet code. The Web server **10** is third-party software that delivers Web pages to a Browser **120**. The Browser **120** views Web pages that are sent from the Web server **110**. Modified HTML with system created images **230** are a final result of the system. Modified HTML **230** is a standard HTML document without proprietary embedded tags and with standard Web graphics.

The System.

A preferred embodiment of the system **100** is provided.

HTML parsing subsystem **610** parses through an HTML document and searches for proprietary tags. If it finds a proprietary tag it hands it to a media caching subsystem **620** for further processing. The media caching subsystem **620** returns a standard HTML tag. The HTML parsing subsystem **610** then replaces the proprietary tag it found with the returned tag. The parsing subsystem **610** then continues searching for a next proprietary tag, repeating the process herein above. The process is finished when no more proprietary tags can be found.

The media caching subsystem **620** determines if an image has been created for the requested proprietary tag. If the image has already been created and the files that built that image have not been modified, the media caching subsystem **620** returns an HTML tag that refers to a previously-generated image. If the image has not been created, the media caching subsystem **620** hands the HTML tag to a media creation subsystem **630**. The media creation subsystem **630** returns an image to the media caching subsystem **620**. The media caching subsystem **620** adds the created image and the HTML tag to a media cache database **640**.

The media cache database **640** contains references to the created images **645**. In a preferred embodiment, the references are the script used to create the image, the names of the images used to create the image, the dates of those files, and the HTML that represents the created image. The media caching subsystem **620** performs lookups in this database to determine if the image has been created. If the image has not been created the media caching subsystem **620** calls upon the media creation subsystem **630** to create the image and then store the results in the media cache database **640**.

The media creation subsystem **630** takes a proprietary tag from the media caching subsystem **620** and generates an image. The image is generated by deciphering the tag and handing it to the media processing engine **650**. After the image is created, the media creation subsystem returns the name of the newly created image to the media caching subsystem **620**.

The media processing engine **650** interprets the proprietary tag and generates the image. The media processing engine **650** looks up images in a media repository to obtain the location of the original file.

The media repository **660** contains original images **665** used in the system **100**.

FIG. **7** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention. An original media **200** is created. The media **200** is placed into the system **100** in the media repository **660**. Similarly, an HTML document with proprietary tags **300** is created and placed on a Web server **110**. A user requests a Web page from a Web browser **120**. The Web server **110** passes the requested page to an HTML parser **610**. The HTML parser **610** parses HTML looking for media tags. The parser **610** looks, up media tags in a media tags database **640**. If the media tag is found, then the system **100** produces a modified HTML document **230**. Otherwise, the media creation subsystem **630** uses the media tag to generate a Web media **220**. The generated Web media **220** is placed in a media cache subsystem **620**. The proprietary media tag is converted by a converter **700** to a standard HTML tag that refers to the generated media **220** in cache. The media tag and the HTML equivalent are stored in the media tags database **640**. Media tags are replaced by standard HTML equivalent to provide a modified HTML document **230**. The modified HTML document **230** is delivered to the Web server **110**. The Web server **100** delivers the modified HTML document **230** to the browser **120** via the Internet for a user to view.

FIG. **8** is a flow chart showing an authoring process according to a preferred embodiment of the invention. The process starts **(800)** when a user adds an original graphic to the system **(810)**. The user then creates an HTML document that contains proprietary media tags **(820)**. The user then places the HTML document on a Web server **(830)** and ends the authoring process **(840)**.

FIG. **9** is a flow chart showing an HTML parsing process according to a preferred embodiment of the invention. The process starts **(900)** when a consumer requests a Web page **(910)**. A Web server hands the request of the Web page to the system **(920)**. The system parses the, Web page **(930)**. The system looks for a media tag **(940)**. If found, the system retrieves the HTML equivalent of the media tag **(950)** and replaces the media tag with the HTML equivalent tag **(960)**. The system continues parsing the Web page for tags **(970)** by returning to step **(940)**. When no more tags are found, the system delivers the modified Web page to the Web server **(980)** and therein ends the process **(990)**.

FIG. **10** is a flow chart showing a media creation process according to a preferred embodiment of the invention. The process starts **(1000)** when the system requests an HTML equivalent to a proprietary media tag **(1010)**. The Media tag is combined with bandwidth information **(1020)**. The subsystem checks if the media tag already exists in the media tag database **(1030)**. If it does, the subsystem checks if any of the original assets used to create the media have been changed **(1040)**. If not, then the subsystem retrieves the HTML equivalent tag from the database **(1050)** and returns the HTML equivalent tag to the requesting system **(1060)**. If any of the original assets used to create the media have been changed (**1040**), then the subsystem removes the media tag entry from the media database (**1070**) and creates the media using the media tag (**1080**). The subsystem then stores the media in a media cache (**1090**). The subsystem generates the HTML referring to the generated media (**1100**) and places the media tag and the HTML equivalent in the media tag database (**1110**). The HTML equivalent is returned to the requesting system (**1060**) and the process stops (**1120**).

The differences between using HTML and the proprietary tags disclosed herein are noted. HTML allows Web designers to create Web page layouts. HTML offers some control of the images. HTML allows the Web designer to set the height and width of an image. However, all of the other image operations disclosed herein are supported by the claimed invention and are not supported by HTML.

Table A herein below provides the claimed proprietary tags according to a preferred embodiment of the invention. The use of the term “freeride” refers to an internal code name for the invention.

TABLE A

Tags
Generate image <freerideimage> mediascript </freerideimage> Generate a standard Web image.
Generate thumbnail image linked to full image <freerideimagethumbnail> mediascript <xs=size ys=size/freerideimagethumbnail> Generate a thumbnail of specified size and link it to the full size version.
Generate zoom and pan image <freerideimagezoom> mediascript </freerideimagezoom> Generate a zoomable/panable image.
Security <freerideimagesecure> </freerideimagesecure> Specifies that all images found between these tags are secured images and the system will determine access before generating.

Table B herein below provides the claimed script commands according to a preferred embodiment of the invention. Additional commands may be added as needed.

TABLE B

Media processing script commands
Add Noise Noise_AddNoise([amount=<value 1 . . . 999>] [gaussian] [grayscale]) This command adds noise to the image.
Adjust HSB AdjustHsb([hue @ <value ±255>] [saturation @ <value ±255>] [brightness @ <value ±255>]) This command allows the HSB of an image to be altered. This can be applied to images of all supported bit-depths.
Adjust RGB AdjustRgb([brightness @ <value ±255>] [contrast @ <value ±255>] [red @ <value ±255>] [green @ <value ±255>] [blue @ <value ±255>] [noclip @ <true, false>] [invert @ <true, false>]) This command allows the contrast, brightness, and color balance of an image to be altered.
Blur Blur(radius @ <value 0.30>) This command applies a simple blur filter on the image.
Blur Convolve Blu_Blur( ) This command commands perform a simple 3×3 convolution for blurring.
Blur Convolve More Blur_MoreBlur( ) This command commands perform a stronger 3×3 convolution for blurring.
Blur Gaussian Blur_GaussianBlur([radius=<value 0.1 . . . 250>]) This command applies a Gaussian blur to the image.
Blur Motion Blur_MotionBlur([distance=<value 1 . . . 250>] [angle=<degrees>]) This command applies motion blurring to the image using the specified distance and angle.
Brush Composite Composite(source @ {<User-Defined Media Object name>} [x @ <pixel>] [y @ <pixel>] [onto] [opacity @ <value 0 . . . 255>] [color @ <color in hexadecimal>] [colorize @ <true, false>] [saturation @ <value 0 . . . 255>]) This command composites the specified “brush” (foreground) image onto the current “target” (background) image.
Colorize Colorize(color @ <color in hexadecimal> [saturation @ <value 0 . . . 255>]) This command changes the hue of the pixels in the image to the specified color.
Convert Convert(rtype @ <bit-depth> {dither @ <value 0 . . . 10>]) This command converts the image to the specified type/bit-depth.
Convolve Convolve(Filter @ <filtername>) This command applies a basic convolution filter to the image. In a user interface driven system, the filters could be stored in files and edited/created by the user.
Crop/Resize Canvas Crop([xs @ {<pixels>, <percentage + “%”>}] [ys @ {<pixels>, <percentage + “%”>}] [xo @ <left pixel>] [yo @ <top pixel>] [padcolor @ <color in hexadecimal>] [padindex @ <value 0 . . . 255>]) This command crops the media to a specified size.
Discard Discard( ) This command removes the designated Media Object from memory.
Drop Shadow DropShadow([dx @ <pixels>] [dy @ <pixels>] [color @ <color in hexadecimal>] [opacity @ <value 0 . . . 255>] [blur @ <value 0 . . . 30>] [enlarge @ <true, false>]) This command adds a drop shadow to the image based on its alpha channel.
Equal Equal(source @ {<User-Defined Media Object name>}) This command compares the current media with the one specified. If the media are different in any way, an error value is returned.
Equalize Equalize([brightness @ <−1, 0 . . . 20>] [saturation @ <−1, 0 . . . 20>]) This command equalizes the relevant components of the media. Equalization takes the used range of a component and expands it to fill the available range.
Export Channel ExportGun(Channel @ <channelname>) This command exports a single channel of the source as a grayscale image.
Find Edges Stylize_FindEdges([threshold=<value 0 . . . 255>] [grayscale] [mono] [invert]) This command finds the edges of the image based on the specified threshold value.
Fix Alpha FixAlpha( ) This command adjusts the RGB components of an image relative to its alpha channel.
Flip Flip(<horizontal, vertical> @ <true, false>) This command flips the media vertically or horizontally.
Frame Add FrameAdd(Source @ <filename>) This command adds the given frame(s) to the specified Media Object.
Glow/Halo

TABLE B-continued

Media processing script commands
Glow(Size @ <value 0 . . . 30> [halo @ <value 0 . . . size>] [color @ <color in hexadecimal>] [opacity @ <value 0 . . . 255>] [blur @ <value 0 . . . 30>] [enlarge @ <true, false>]) This command produces a glow or halo around the image based on the image's alpha.
High Pass Other__HighPass([radius=<value 0.1 . . . 250>]) This command replaces each pixel with the difference between the original pixel and a Gaussian blurred version of the image.
Import Channel ImportGun(channel @ <channel name> source @ {<User-Defined Media Object name>} [rtype @ <bit-depth>]) This command imports the specified source image (treated as a grayscale) and replaces the selected channel in the original.
Load Load(Name @ <filename> [type @ <typename>] [transform @ <true, false>]) This command loads a media from the specified file.
Maximum Other__Maximum([radius=<value 1 . . . 10>]) This command scans the area specified by the radius surrounding each pixel, and then replaces the pixel with the brightest pixel found.
Minimum Other__Minimum([radius=<value 1 . . . 10>]) This command scans the area specified by the radius surrounding each pixel, and then replaces the pixel with the darkest pixel found.
Normalize Normalize([clip @ <value 0 . . . 20>]) This command expands the volume of the sample to the maximum possible.
Pixellate Mosaic Pixellate__Mosaic([size=<value 2 . . . 64>]) This command converts the image to squares of the specified size, where each square contains the average color for that part of the image.
Pixellate Fragment Pixellate__Fragment([radius=<value 1 . . . 16>]) This command produces four copies of the image displaced in each direction (up, down, left, right) by the specified radius distance and then averages them together.
Quad Warp Quadwarp([tlx=<position>] [tly=<position>] [trx=<position>] [try=<position>] [blx=<position>] [bly=<position>] [brx=<position>] [bry=<position>] [smooth]) This command takes the corners of the source image and moves them to the specified locations, producing a warped effect on the image.
Reduce to Palette Reduce([colors @ <num colors>] [netscape @ >true, false>] [b&w @ <true, false>] [dither @ <value 0 . . . 10>] [dithertop @ <value 0 . . . 10>] [notbackcolor] [pad @ <true, false>]) This command applies a specified or generated palette to the image.
Rotate Rotate(Angle @ <value 0 . . . 359> [smooth @ <true, false>] [enlarge @ <true, false>] [xs @ <pixels>] [ys @ <pixels>]) This command rotates the media by the specified angle in degrees.
Rotate 3D Rotate3d([anglex @ <angle ±89>] [angley @ <angle ±89>] [distance @ <value>]) This command rotates the image in 3D about either the x-axis or y-axis.
Save Save([type @ <image-type>]) This command saves a media to the specified file.
Scale Scale([xs @ {<pixels>, <percentage + "%">}] [ys @ {<pixels>, <percentage + "%">}] [constrain @ <true, false>] [alg @ {"fast", "smooth", "outline"}] [x1 @ <pixels>] [y1 @ <pixels>] [x2 @ <pixels>] [y2 @ <pixels>]) This command scales the image to the specified size.
Select Selection([source @ <User-Defined media Object>}] [remove @ <true, false>] [invert @ <true, false>] [backcolor] [color=<color>] [index=<value>] [opacity @ <value 0 . . . 255>]) This command manages the selected region for the current Media Object.
Set Color SetColor([backcolor @ <color in hexadecimal>] [forecolor @ <color in hexadecimal>] [backindex @ <value 0 . . . 255>] [foreindex @ <value 0 . . . 255>] [transparency @ ("on", "off")]) This command allows the background color, foreground color, and transparency state of an image to be set.
Set Resolution SetResolution([dpi @ <value>] [xdpi @ <value>] [ydpi @ <value>]) This command changes the DPI of the image in memory.
Sharpen Sharpen__Sharpen( ) This command sharpens the image by enhancing the high-frequency component of the image.
Sharpen More Sharpen__SharpenMore( ) This command sharpens the image by enhancing the high-frequency component of the image, but is stronger than the standard sharpening.
Stylize Diffuse Stylize__Diffuse([radius=<value 0 . . . >] [lighten] [darken] This command diffuses the image by randomizing the pixels within a given pixel radius.
Stylize Embose Stylize__Emboss([height=<value 1 . . . 10>] [angle=<degrees>] [amount=<percentage 1 . . . 500>]) This command converts the image to an embossed version.
Text Drawing DrawText(Text @ <string> Font @ <font file> [size @ <value>] [color @ <color in hexadecimal>] [smooth @ <true, false>] [<left, right, top, bottom> @ <true, false>] [x @ <pixel>] [y @ <pixel>] [wrap @ <pixel-width>] [justify @ {left, center, right}] [angle @ <angle >]) This command composites the specified text string onto the image.
Text Making MakeText(text @ <string> font @ <font file> [path @ <path to font directory>] [size @ <value 1 . . . 4095>] [color @ <color in hexadecimal>] [smooth @ <true, false>] [wrap @ <pixel-width>] [justify @ {left, center, right}] [angle @ <angle>]) This command creates a new image that includes only the specified text.
Trace Contour Stylize__TraceContour([level=<value 0 . . . 255>] [upper] [invert]) This command traces the contour of the image at the specified level (for each gun).
Unsharpen Mask Sharpen__UnsharpMask([amount=<percentage 1 . . . 500>] [radius=<value 0.1 . . . 250>] [threshold=<value 0 . . . 255>]) This command enhances the edges and detail of an image by exaggerating differences between the image and a gaussian blurred version of the same image.
Zoom Zoom([xs @ <pixels>] [ys @ <pixels>] [scale @ <value>] [x @ <left pixel>] [y @ <top pixel>]) This command zooms in on a specified portion of the media and fits it to the specified size. This constitutes a crop followed by a scale.

Table C herein below provides a list of features provided by a preferred embodiment of the invention. It is noted that the list of features included in Table C is by no means complete. In other embodiments, the list of features is expanded or reduced as needed.

TABLE C

System Feature List
Reads and writes various file formats
BMP, GIF, JPG, PNG, TIF, PICT, TGA, PSD, FPX
Supports many image processing operations
Dynamically creates Web images from original assets
Dynamically creates thumbnail images
Dynamically creates images that can be panned and zoomed without browser plug-ins or special file formats
Automatically propagates changes of original assets throughout a Web site
Uses an intelligent caching mechanism
Clean up image cache on demand
Eliminates orphaned image files
Optimizes Web server cache by providing most recent images
Renders TrueType fonts on the server instead of browser
Uses intelligent scaling of line drawings
Allows Web designers to manipulate images with proprietary tags
Preserves original image assets
Optimizes Web server traffic by adiusting the bandwidth of graphics
Optimizes images for client connection speed
Allows clients to specify the quality of images on a Web site
Allows Web designers to dynamically create images by manipulating proprietary tags in their applications (server or client side)

FIG. **11** is a screen shot showing an administration tool according to a preferred embodiment of the invention. Specifically, FIG. **11** shows an administration page that contains cached images of generated scripts. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **12** displays a structure of a database record used for the system according to a preferred embodiment of the invention. A Script Table **1200** has 5 columns, Media Script **1210**, HTML Equivalent **1220**, Bandwidth **1230**, Generated File **1240**, and Dependency List **1250**. A Dependency Table **1260** has two columns, File Name **1270** and Modification Date **1280**.

Snowboard Store Example

Background.

The snowboard store highlights several features of the claimed system. The snowboard store is an imaginary store that allows a user to configure his or her snowboard. The store consists of five logos, five board colors, and four boards. The consumer dicks on the buttons to change the snowboard represented in the middle of the screen. When the consumer has configured the snowboard they the snowboard can be purchased by selecting a buy button.

Prior Art Method.

To create the snowboard site today, the Web designer must render all possible combinations of the board. The number of combinations is five logos × five board colors × four boards=100. The designer also must render all the buttons. The creation process is very tedious and involves a lot of production work. Typically, most Web sites do not even attempt such an endeavor. Also, other issues must be addressed, such as, for example, updating the Web site and scripting. For example, updating a single logo involves updating a minimum of 20 images.

The prior art method sustains a graphic intensive site that requires management of at least 100 images. Updates to the Web site are time-consuming and prone to human error.

The Claimed Method.

A preferred embodiment of the method scripts the image creation process in HTML to create a dynamic Web site. There is no need to create over 100 images. The claimed system generates images on demand. The Web site only needs to create original assets. The scripting process involves writing the proprietary scripts. In the current example herein, scripting buttons is very simple. Once one button is created, simply copy and paste the HTML to create another button or many buttons. Only the name of the image to be overlaid on the button must be changed. The Webmaster then creates a simple program that reads what object a user has clicked on and generates a proprietary tag. The tag is then sent to the claimed system to generate a center image.

The claimed method allows the creation of all 100 combinations automatically. When the Web site receives an updated image, only the original image needs to be updated. Any change to the original image automatically propagates throughout the system. The Web site is easier to manage. Testing of the Web site is easier because there is no need to test all 100 combinations. A small subset of combinations will guarantee adequate coverage.

Processing of an Image Tag Example (FIG. **13**–**16**).

FIG. **13** shows two original images **1300** and **1310** to be processed according to a preferred embodiment of the invention.

FIG. **14** shows a portion on an HTML document with a proprietary tag **1400**, <freerideimage></freerideimage> according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **15** shows an HTML document **1500** as viewed in a browser and an HTML document source **1510**, according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **16** shows a generated GIF image **1600** according to a preferred embodiment of the invention.

Accordingly, although the invention has been described in detail with reference to a particular preferred embodiment, persons possessing ordinary skill in the art to which this invention pertains will appreciate that various modifications and enhancements may be made without departing from the spirit and scope of the claims that follow.

What is claimed is:

1. A process for delivering an original media, comprising the steps of:

placing said original media in a network system;

creating an HTML document or browser language having proprietary media tags and placing said HTML document or browser language onto a Web server;

actuating a user request from a customer browser for a Web page by having said Web server pass said requested Web page to an HTML parser on said network system;

parsing said HTML document or browser language on said parser by looking for said proprietary media tags;

looking up said proprietary media tags in a media tags database on said network system;

if at least one media tag of said proprietary media tags is not found,

generating a Web media by using said at least one media tag; placing said

generated Web media in a media cache on said network system;

converting said at least one media tag to at least one standard HTML equivalent tag that refers to said media in said cache; and

storing said at least one media tag and said at least one standard HTML equivalent tag in said database;

modifying said HTML document or browser language by replacing said at least one media tag by said at least one standard HTML equivalent tag;

delivering said modified HTML document or browser language from said network system to said Web server; and

delivering said modified HTML document or browser language from said Web server to said customer browser for said customer to view.

2. A network system for delivering an original media, comprising:

means for placing said original media in said network system;

means for creating HTML or browser language having proprietary media tags and placing said HTML or browser language onto a Web server;

means for actuating a request from a customer browser for a Web page by having said Web server pass said requested Web page to an HTML parser on said network system;

means for parsing said HTML document on said parser by looking for said proprietary media tags;

means for looking up said proprietary media tags in a media tags database on said network system;

if at least one media tag of said proprietary media tags is not found,

means for generating a Web media by using said at least one media tag;

placing said generated Web media in a media cache on said network system;

means for converting said at least one media tag to at least one standard HTML equivalent tag that refers to said media in said cache; and

means for storing said at least one media tag and said at least one standard HTML equivalent tag in said database;

means for generating HTML or browser language by replacing said at least one media tag by said at least one standard HTML equivalent tag;

means for delivering said generated HTML or browser language from said network system to said Web server; and

means for delivering said modified generated HTML or browser language from said Web server to said customer browser for said customer to view.

3. An media delivery system using a Java servlet engine, a Web server, and a customer browser for generating HTML or browser language, said generated HTML or browser language having a proprietary image, from an original HTML document or browser language embedded with a proprietary tag, comprising:

an HTML parser subsystem adapted to search for a proprietary tag in said original HTML or browser language and to replace said proprietary tag with a standard HTML tag;

a media caching subsystem adapted to determine a status of existing or needs to be created of said proprietary image for said proprietary tag, and to send said standard HTML tag to said media creation subsystem;

a media cache database adapted to store data associated with said proprietary image;

a media creation subsystem adapted to decipher said proprietary tag;

a media processing engine adapted to receive from said media creation subsystem said deciphered proprietary tag and to interpret said tag to generate said proprietary media; and

a media repository adapted to store at least one original media associated with said proprietary media;

wherein said media cache database further comprises:

a script table having a media processing script column, an HTML Equivalent column, a Bandwidth column, a Generated File column, and a Dependency List column; and

a dependency table having a File Name column and a Modification Date column.

4. The system of claim **3**, wherein said original HTML or browser language is embedded with a plurality of proprietary tags, said proprietary tags associated with a plurality of proprietary media and associated with a plurality of original media.

5. The system of claim **3**, wherein said stored data associated with said proprietary media further comprises:

a script used to create said proprietary media, said script having an associated date;

a name of said at least one original media used to create said proprietary media, said name having an associated date; and

said generated HTML or browser language.

6. The system of claim **5**, wherein said media caching subsystem further comprises:

means for determining if said associated script data and said associated original media date have been modified.

7. The system of claim **3**, wherein said HTML parser subsystem further comprises:

means for sending said proprietary tag to said media caching subsystem for further processing;

means for receiving from said media caching subsystem said standard HTML tag;

means for searching for a next proprietary tag to replace said next proprietary tag with a next standard HTML tag; and

means for determining that no more proprietary tags exist in said original HTML document or browser language.

8. The system of claim **3**, wherein said media caching subsystem further comprises:

means for determining if said proprietary media has been modified.

9. A process for creating a media associated with an HTML equivalent tag and associated with a media tag for a system, comprising the steps of:

receiving a request from said system for said HTML equivalent tag;

combining said media tag with bandwidth information;

determining if a media tag entry associated with said media tag exists in a media tag database;

if said media tag entry exists in said media database,

determining if at least one original asset used to create said media has changed;

if said at least one original asset used to create said media has not changed,

retrieving said HTML equivalent tag from said database; and

return said HTML equivalent tag to said system;

if said at least one original asset used to create said media has changed,

removing said media tag entry from said database;

create said media using said media tag;

storing said media in a media cache;

generating an HTML document referring to said generated media;

placing said media tag and said HTML equivalent tag in said media database; and

return said HTML equivalent tag to said system;

if said media tag entry does not exist in said media database,

create said media using said media tag;

storing said media in a media cache;

generating HTML or browser language referring to said generated media;

placing said media tag and said HTML equivalent tag in said media database; and

return said HTML equivalent tag to said system.

* * * * *

Exhibit B

US006964009B2

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

(10) **Patent No.: US 6,964,009 B2**
(45) **Date of Patent: Nov. 8, 2005**

(54) **AUTOMATED MEDIA DELIVERY SYSTEM**

(75) Inventors: **Christopher Samaniego**, San Francisco, CA (US); **Nelson H. Rocky Offner**, Kensington, CA (US); **Adrian D. Thewlis**, Sausalito, CA (US); **David R. Boyd**, San Francisco, CA (US); **David C. Salmon**, San Rafael, CA (US); **Joshua N. Devan**, Kentfield, CA (US)

(73) Assignee: **Automated Media Processing Solutions, Inc.**, Mill Valley, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 754 days.

(21) Appl. No.: **09/929,904**

(22) Filed: **Aug. 14, 2001**

(65) **Prior Publication Data**

US 2002/0078093 A1 Jun. 20, 2002

Related U.S. Application Data

(63) Continuation-in-part of application No. 09/425,326, filed on Oct. 21, 1999.

(60) Provisional application No. 60/226,043, filed on Aug. 16, 2000.

(51) **Int. Cl.**^{7} **G06F 15/00**; G06F 17/00; G06F 15/16

(52) **U.S. Cl.** **715/501.1**; 715/513; 715/517; 709/203

(58) **Field of Search** 715/501.1, 517, 715/513; 709/203; 345/629

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,870,552	A	*	2/1999	Dozier et al.	709/219
5,880,740	A	*	3/1999	Halliday et al.	345/629
5,890,170	A	*	3/1999	Sidana	715/501.1
5,895,476	A	*	4/1999	Orr et al.	715/517
5,937,160	A	*	8/1999	Davis et al.	709/203
6,009,436	A	*	12/1999	Motoyama et al.	707/102
6,456,305	B1	*	9/2002	Qureshi et al.	345/800
6,563,517	B1	*	5/2003	Bhagwat et al.	345/735
6,591,280	B2	*	7/2003	Orr	715/513
6,623,529	B1	*	9/2003	Lakritz	715/536

OTHER PUBLICATIONS

Zaiane et al., Mining multimedia data, 1998 ACM Conference of the Center for Advanced Studies on Collaborative research, Nov. 1998, pp. 1–18.*

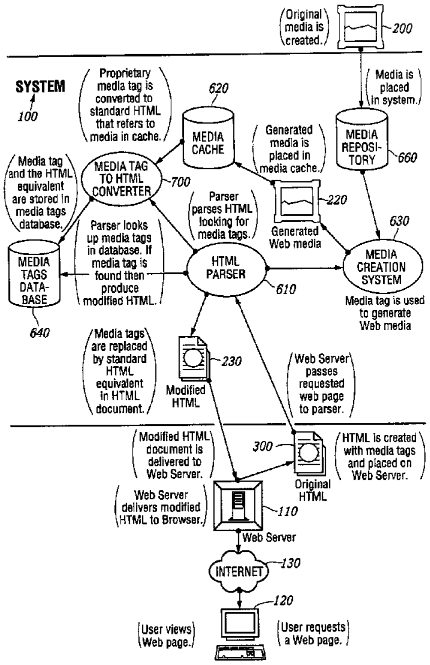
* cited by examiner

Primary Examiner—William Bashore
(74) *Attorney, Agent, or Firm*—Glenn Patent Group; Michael A. Glenn

(57) **ABSTRACT**

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed by an author within URLs embedded within Web documents.

8 Claims, 23 Drawing Sheets


Original media is created.
200
SYSTEM
100
Proprietary media tag is converted to standard HTML that refers to media in cache.
620
MEDIA CACHE
Media is placed in system.
MEDIA REPOSI-TORY
660
Generated media is placed in media cache.
Media tag and the HTML equivalent are stored in media tags database.
MEDIA TAG TO HTML CONVERTER
700
Parser parses HTML looking for media tags.
220
Generated Web media
630
Parser looks up media tags in database. If media tag is found then produce modified HTML.
MEDIA TAGS DATA-BASE
HTML PARSER
610
MEDIA CREATION SYSTEM
Media tag is used to generate Web media
640
Media tags are replaced by standard HTML equivalent in HTML document.
230
Modified HTML
Web Server passes requested web page to parser.
Modified HTML document is delivered to Web Server.
300
HTML is created with media tags and placed on Web Server.
Original HTML
Web Server delivers modified HTML to Browser.
110
Web Server
INTERNET
130
120
User views Web page.
User requests a Web page.

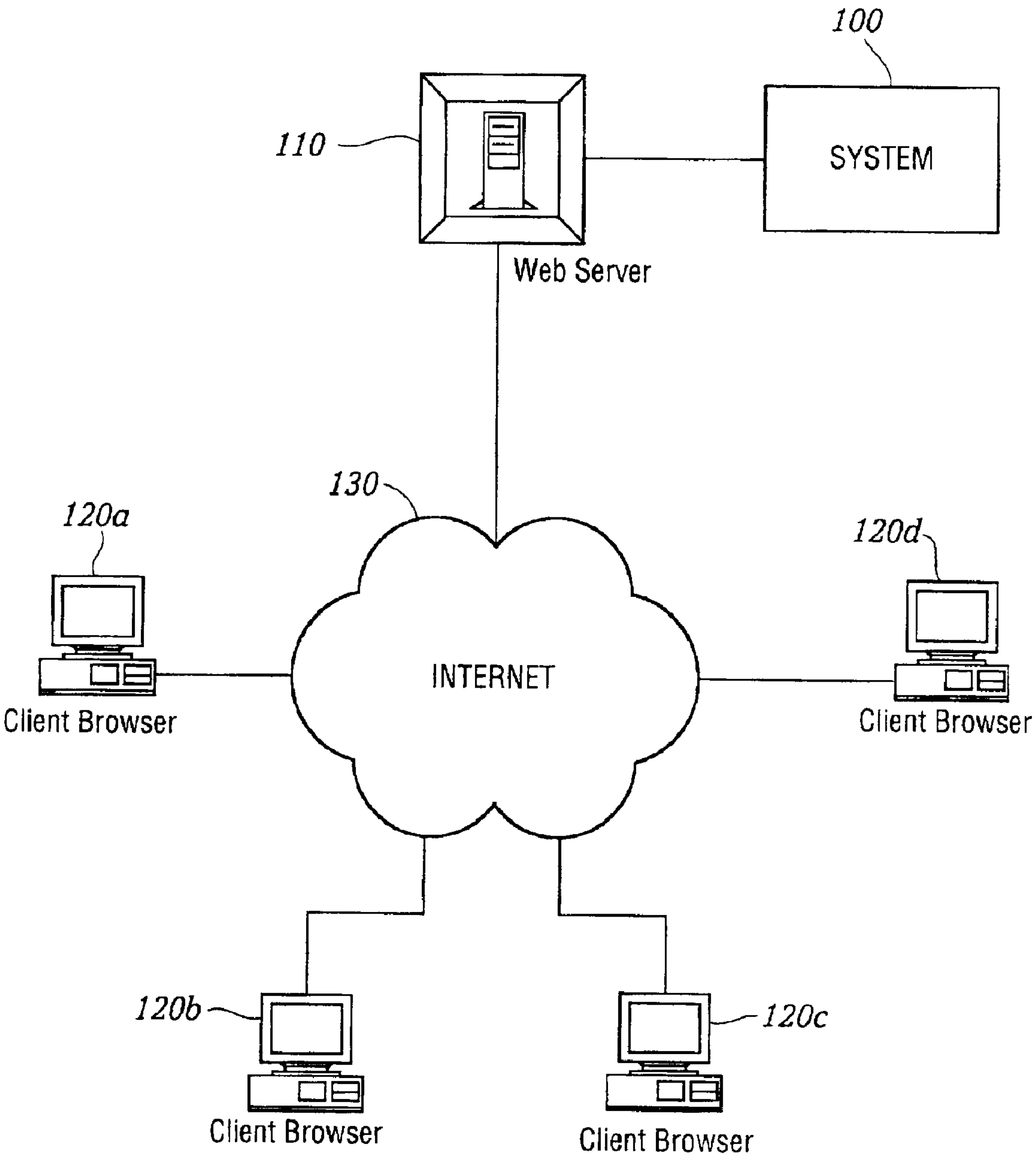

100
110
SYSTEM
Web Server
130
120a
120d
INTERNET
Client Browser
Client Browser
120b
120c
Client Browser
Client Browser

FIG. 1

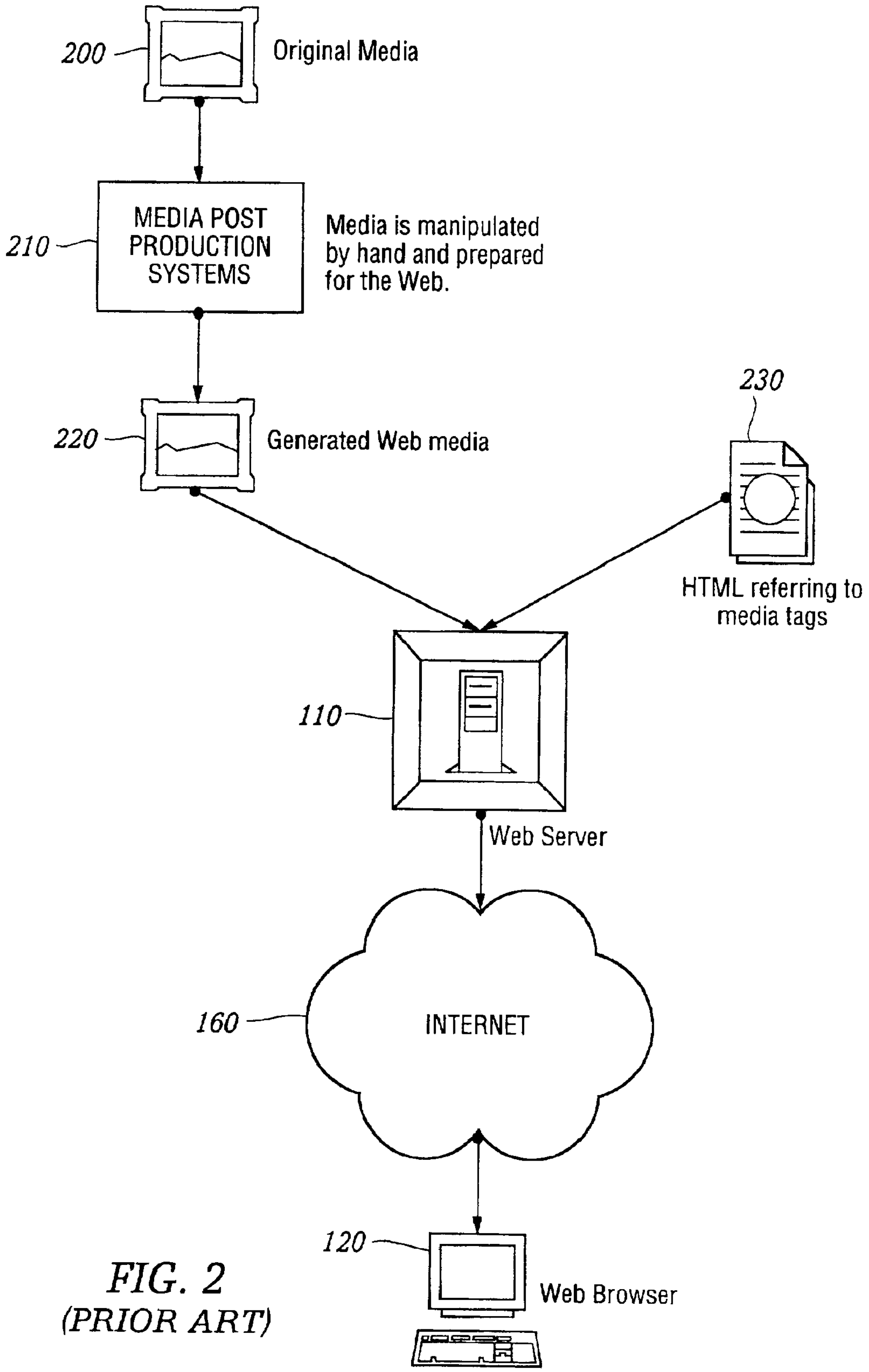

200
Original Media
210
MEDIA POST PRODUCTION SYSTEMS
Media is manipulated by hand and prepared for the Web.
220
Generated Web media
230
HTML referring to media tags
110
Web Server
160
INTERNET
120
Web Browser

FIG. 2
(PRIOR ART)

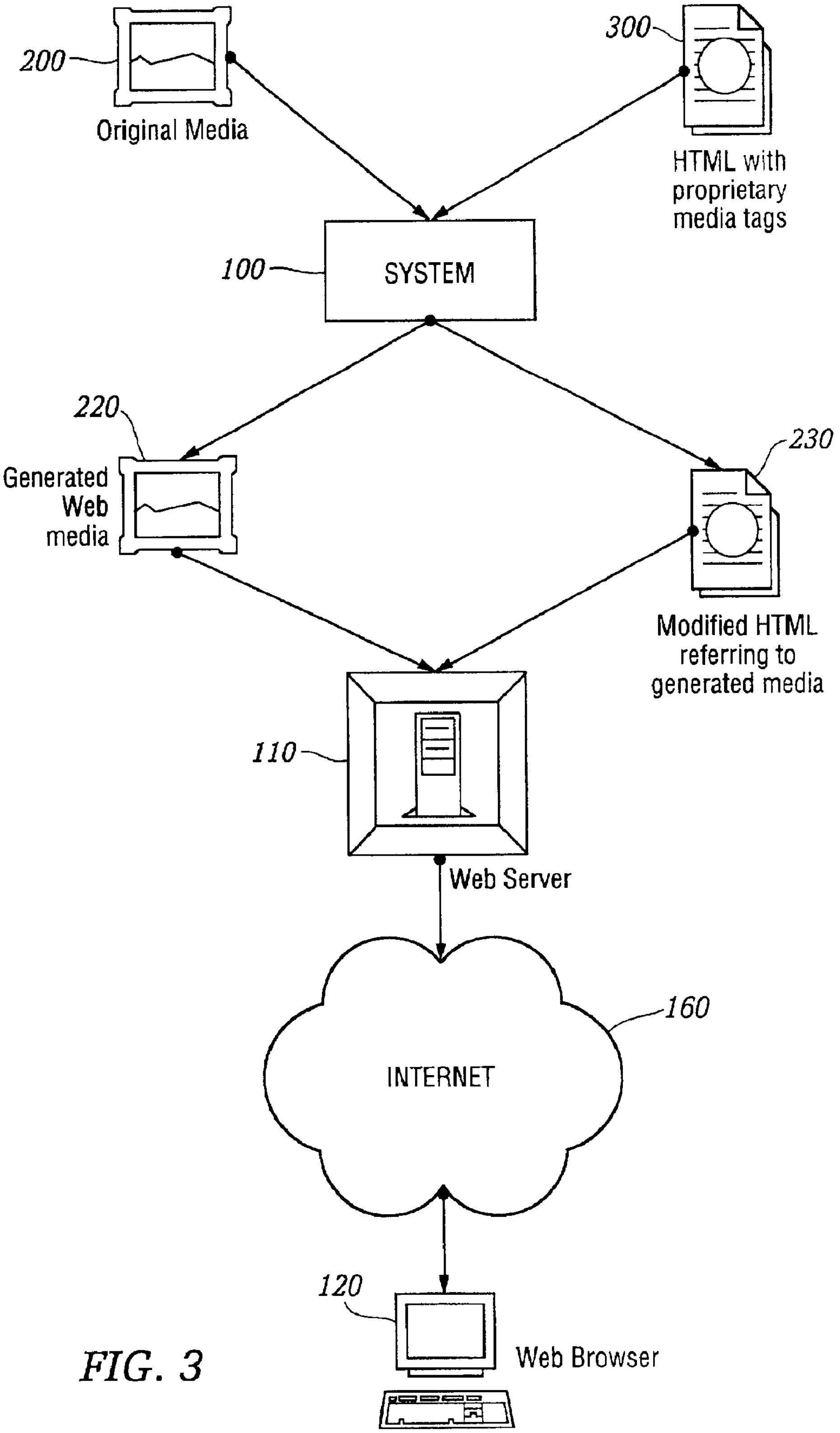

200
Original Media
300
HTML with proprietary media tags
100
SYSTEM
220
Generated Web media
230
Modified HTML referring to generated media
110
Web Server
160
INTERNET
120
Web Browser

FIG. 3

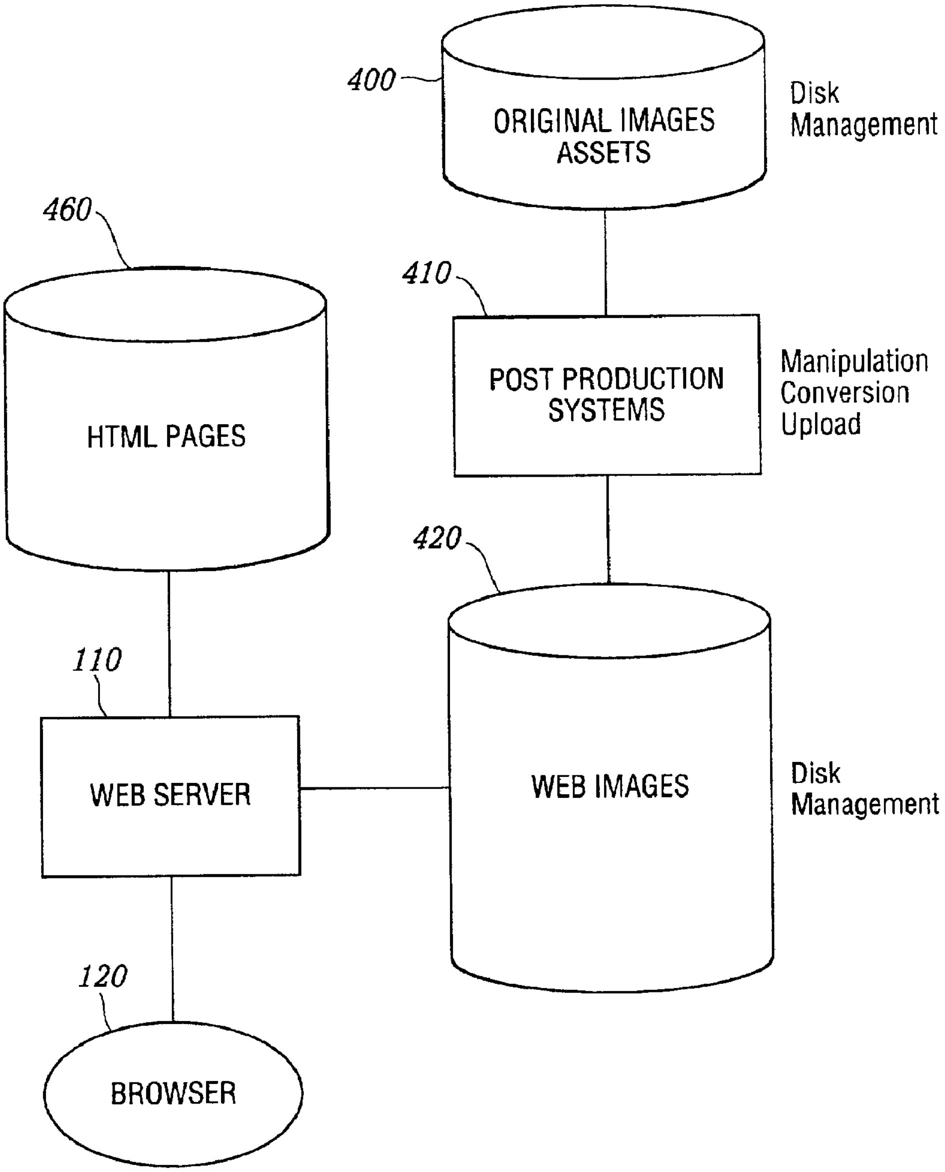


FIG. 4
(PRIOR ART)

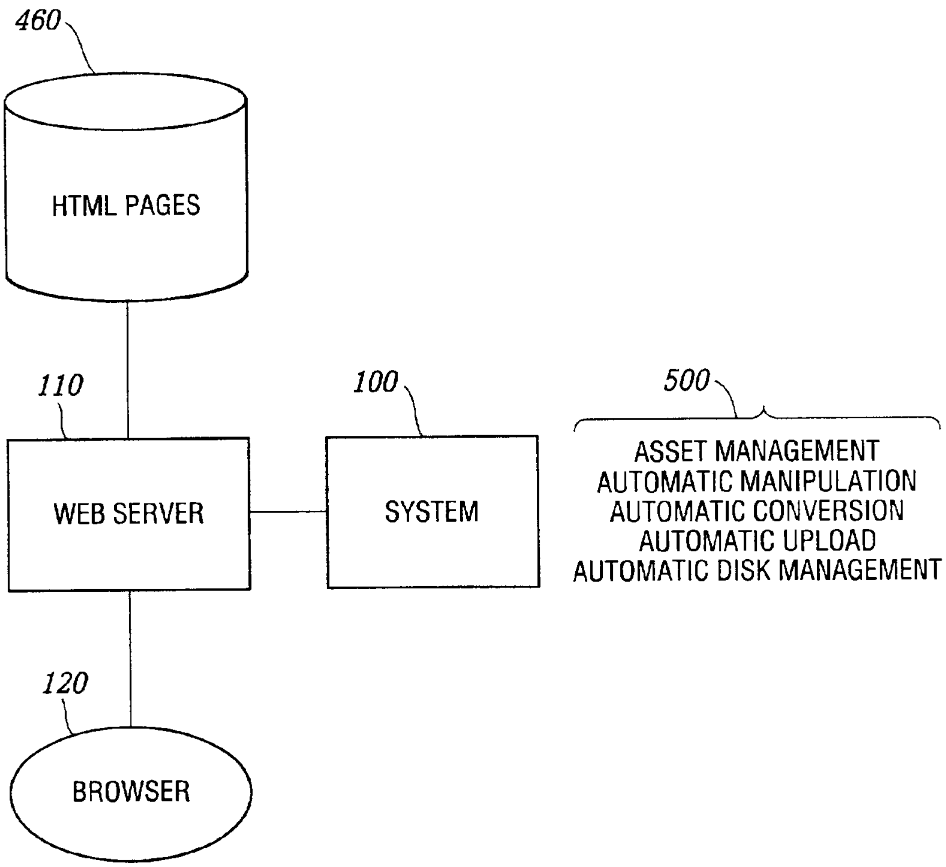

460
HTML PAGES
110
WEB SERVER
100
SYSTEM
500
ASSET MANAGEMENT
AUTOMATIC MANIPULATION
AUTOMATIC CONVERSION
AUTOMATIC UPLOAD
AUTOMATIC DISK MANAGEMENT
120
BROWSER

FIG. 5

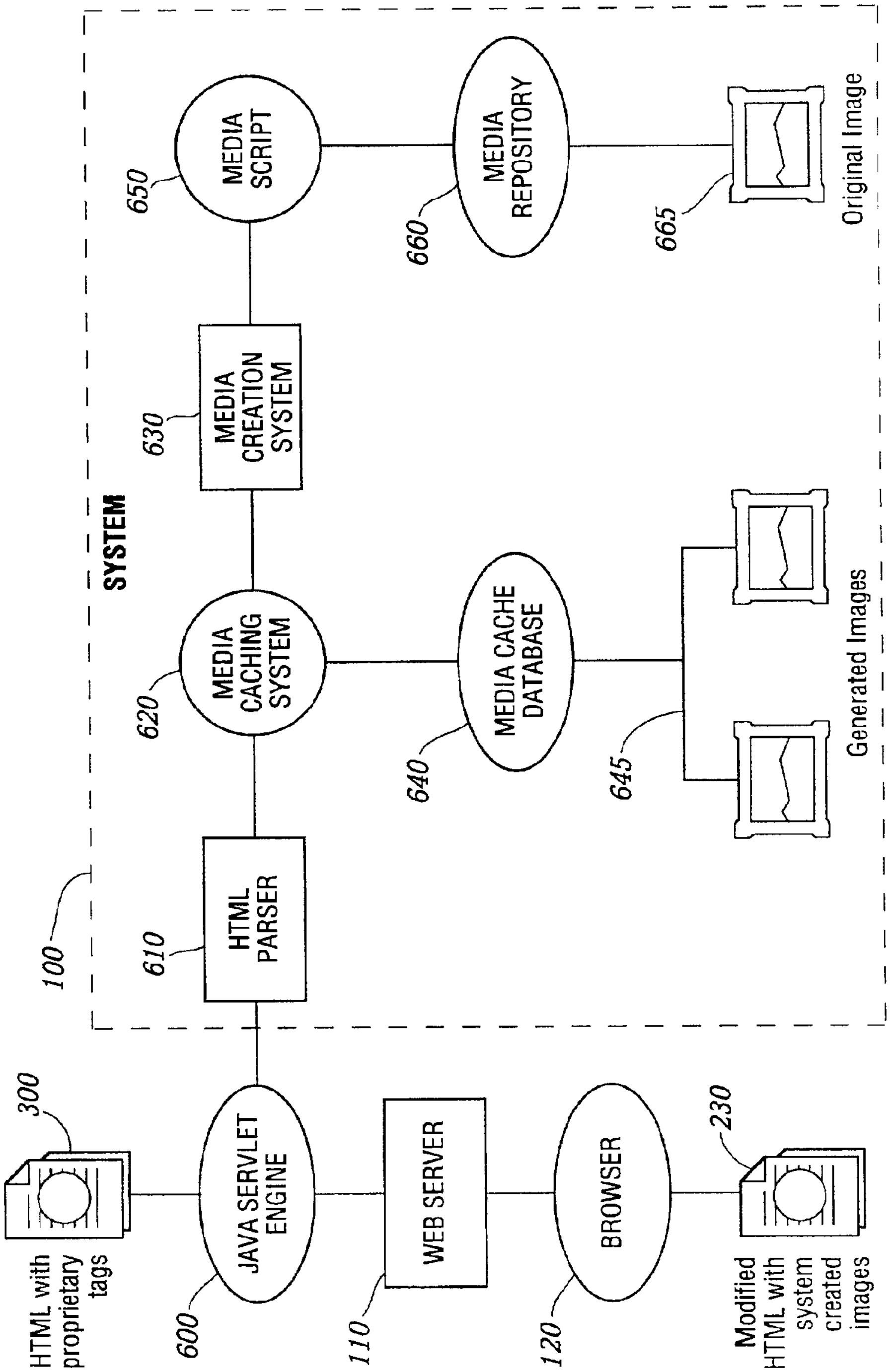

100
SYSTEM
610
HTML PARSER
620
MEDIA CACHING SYSTEM
630
MEDIA CREATION SYSTEM
650
MEDIA SCRIPT
640
MEDIA CACHE DATABASE
660
MEDIA REPOSITORY
645
Generated Images
665
Original Image
300
HTML with proprietary tags
600
JAVA SERVLET ENGINE
110
WEB SERVER
120
BROWSER
230
Modified HTML with system created images

FIG. 6

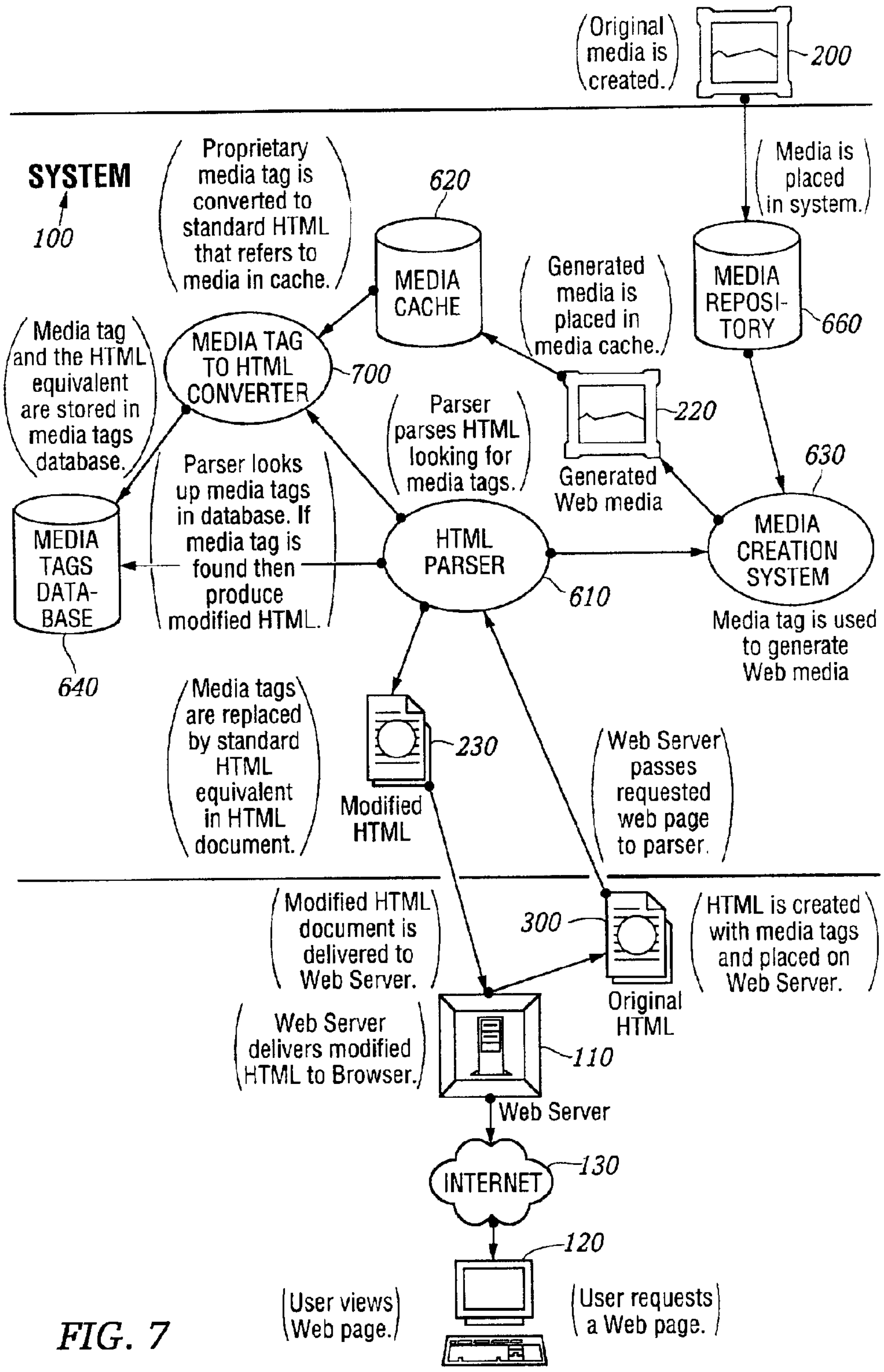

Original media is created.
200
SYSTEM
100
Media is placed in system.
Proprietary media tag is converted to standard HTML that refers to media in cache.
620
MEDIA CACHE
Generated media is placed in media cache.
MEDIA REPOSITORY
660
Media tag and the HTML equivalent are stored in media tags database.
MEDIA TAG TO HTML CONVERTER
700
Parser parses HTML looking for media tags.
220
Generated Web media
630
Parser looks up media tags in database. If media tag is found then produce modified HTML.
MEDIA TAGS DATABASE
HTML PARSER
610
MEDIA CREATION SYSTEM
Media tag is used to generate Web media
640
Media tags are replaced by standard HTML equivalent in HTML document.
230
Modified HTML
Web Server passes requested web page to parser.
Modified HTML document is delivered to Web Server.
300
HTML is created with media tags and placed on Web Server.
Original HTML
Web Server delivers modified HTML to Browser.
110
Web Server
INTERNET
130
120
User views Web page.
User requests a Web page.

FIG. 7

AUTHORING FLOWCHART

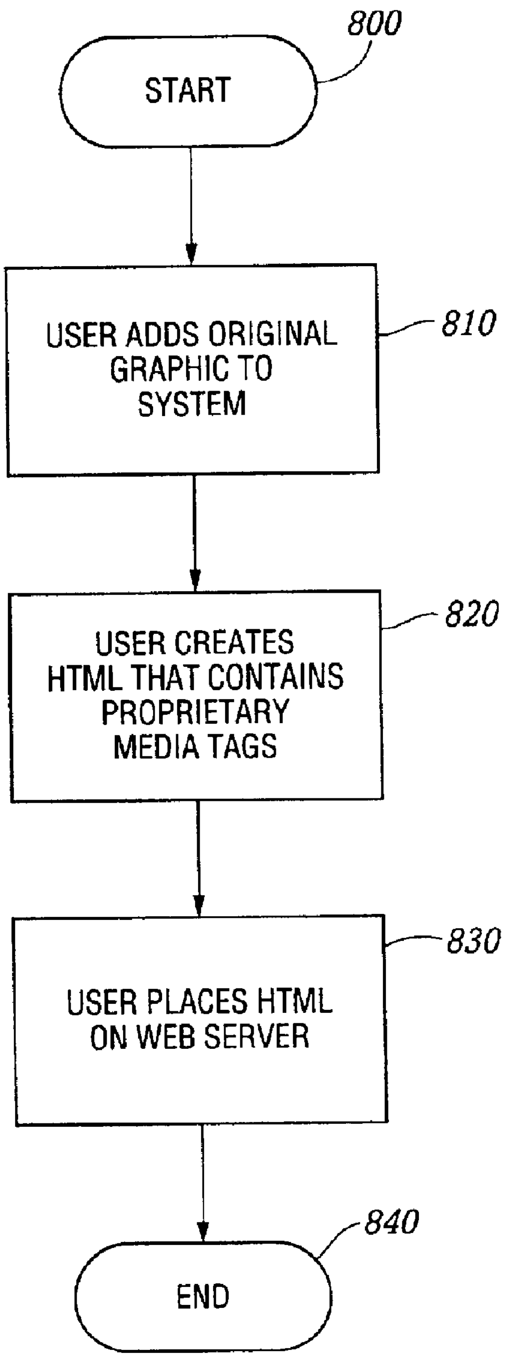

START
800
USER ADDS ORIGINAL GRAPHIC TO SYSTEM
810
USER CREATES HTML THAT CONTAINS PROPRIETARY MEDIA TAGS
820
USER PLACES HTML ON WEB SERVER
830
END
840

FIG. 8

HTML PARSING FLOWCHART

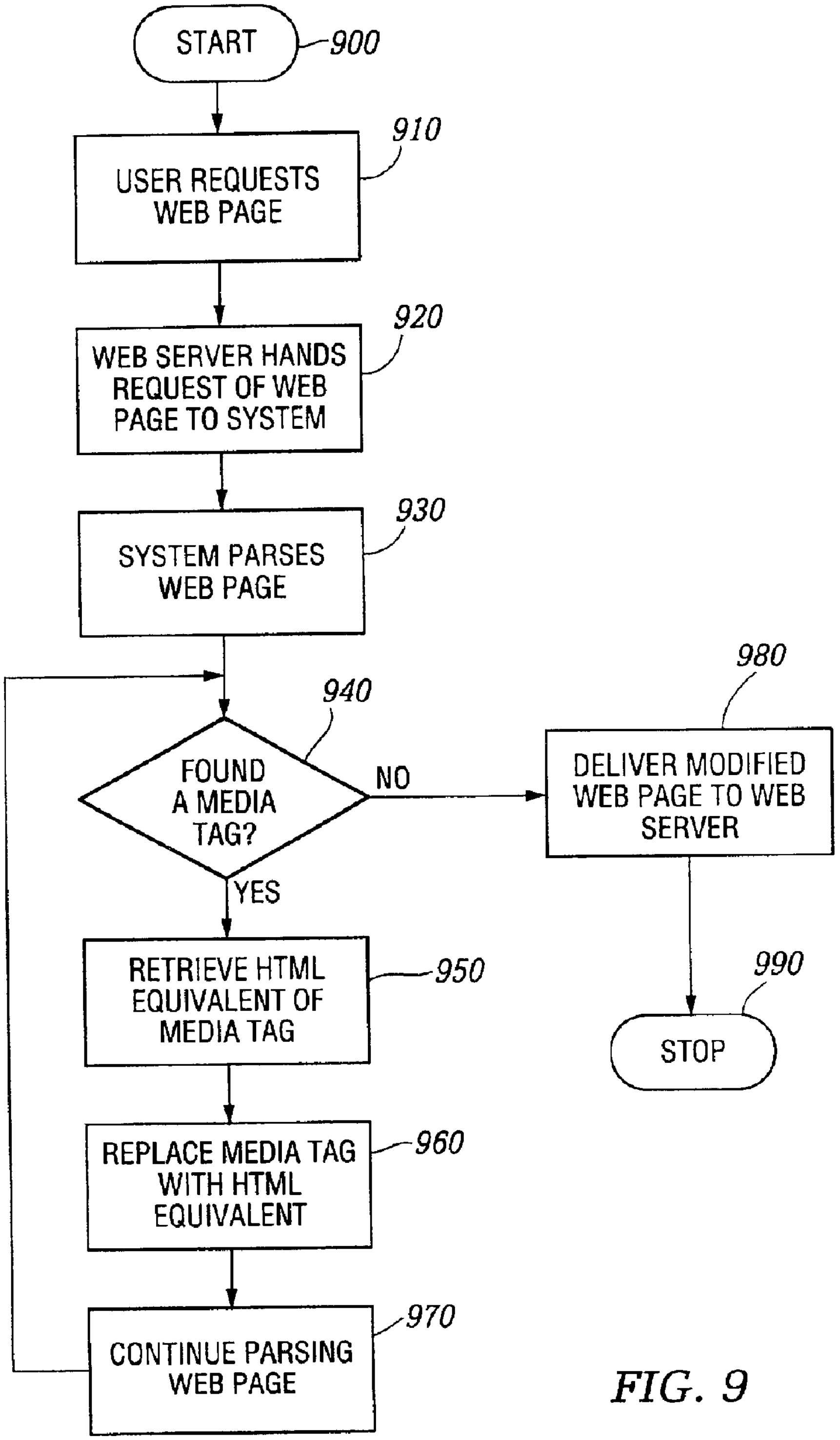

START
900
USER REQUESTS WEB PAGE
910
WEB SERVER HANDS REQUEST OF WEB PAGE TO SYSTEM
920
SYSTEM PARSES WEB PAGE
930
FOUND A MEDIA TAG?
940
NO
YES
DELIVER MODIFIED WEB PAGE TO WEB SERVER
980
STOP
990
RETRIEVE HTML EQUIVALENT OF MEDIA TAG
950
REPLACE MEDIA TAG WITH HTML EQUIVALENT
960
CONTINUE PARSING WEB PAGE
970

FIG. 9

MEDIA CREATION FLOWCHART

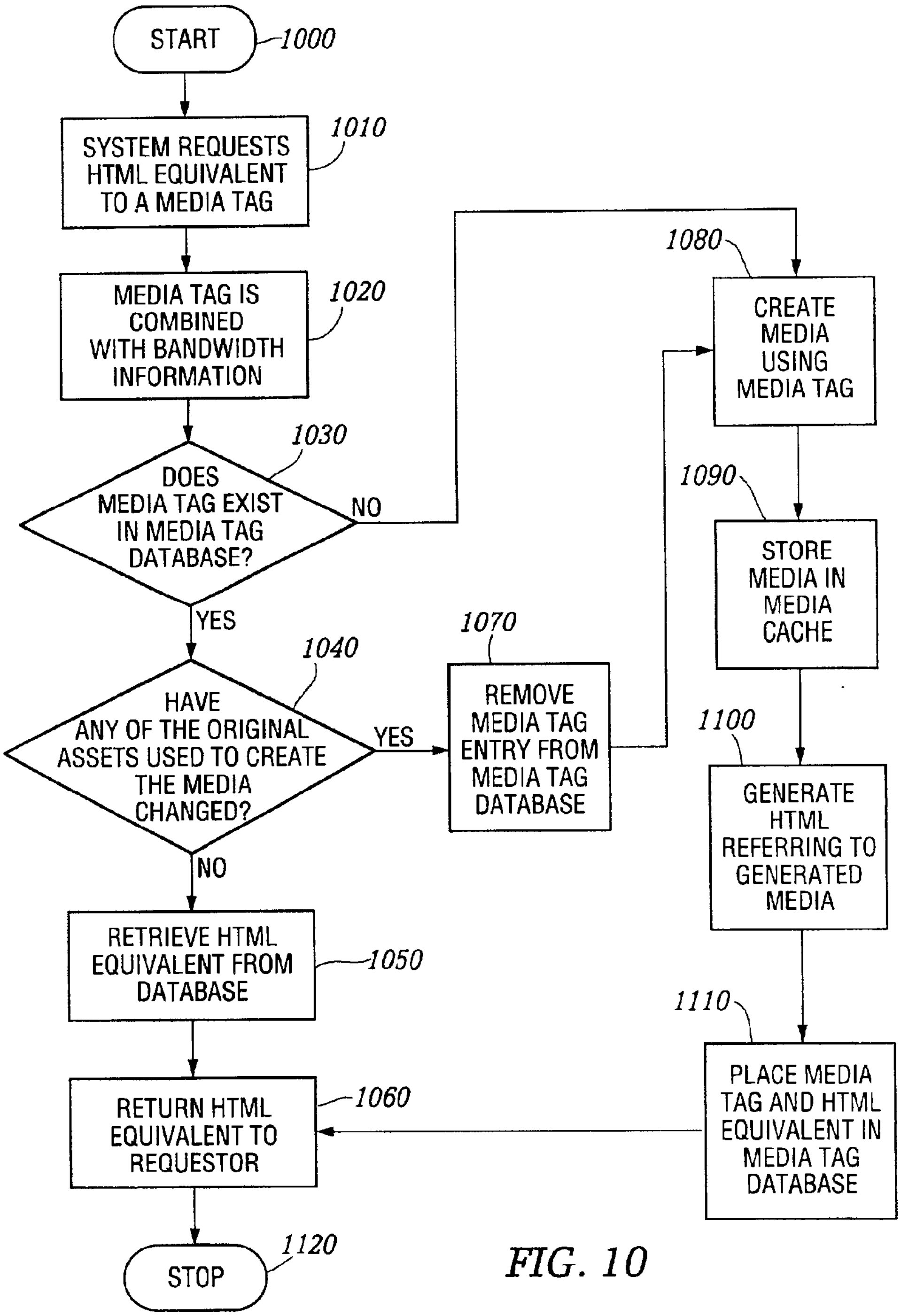
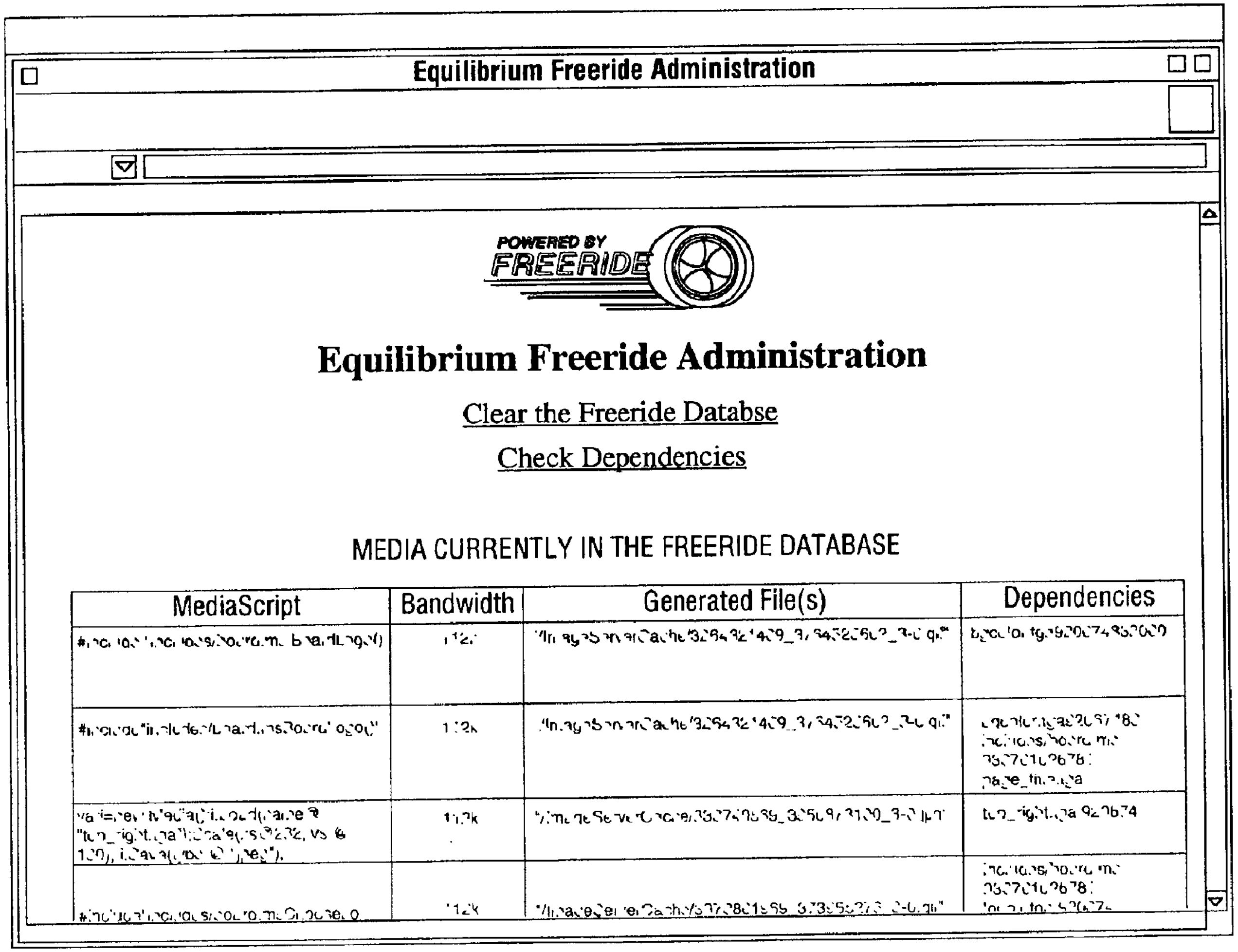

START
1000
SYSTEM REQUESTS HTML EQUIVALENT TO A MEDIA TAG
1010
MEDIA TAG IS COMBINED WITH BANDWIDTH INFORMATION
1020
DOES MEDIA TAG EXIST IN MEDIA TAG DATABASE?
1030
NO
YES
HAVE ANY OF THE ORIGINAL ASSETS USED TO CREATE THE MEDIA CHANGED?
1040
YES
NO
REMOVE MEDIA TAG ENTRY FROM MEDIA TAG DATABASE
1070
CREATE MEDIA USING MEDIA TAG
1080
STORE MEDIA IN MEDIA CACHE
1090
GENERATE HTML REFERRING TO GENERATED MEDIA
1100
PLACE MEDIA TAG AND HTML EQUIVALENT IN MEDIA TAG DATABASE
1110
RETRIEVE HTML EQUIVALENT FROM DATABASE
1050
RETURN HTML EQUIVALENT TO REQUESTOR
1060
STOP
1120

FIG. 10



Equilibrium Freeride Administration

Clear the Freeride Databse

Check Dependencies

MEDIA CURRENTLY IN THE FREERIDE DATABASE

MediaScript	Bandwidth	Generated File(s)	Dependencies
#include "includes/board.ms [illegible]()	112k	"/ImageServerCache/3284321409_3784326502_3-0.gif"	bgcolor.tga 920074852009
#include "includes/boardinsBoard logo()"	112k	"/ImageServerCache/3284321409_3784326502_3-0.gif"	logo.tga 852097180 includes/board.ms 852701029678 page_in.tga
var [illegible] = new Media(); [illegible] load(name = "top_right.tga"); scale(s 232, vs 130); [illegible]("jpeg");	112k	"/ImageServerCache/3327478589_3250973100_3-0.jpg"	top_right.tga 920674
#include "includes/board.ms [illegible]	112k	"/ImageServerCache/5972801589_3735[illegible]_3-0.gif"	includes/board.ms 852701029678 [illegible]

FIG. 11

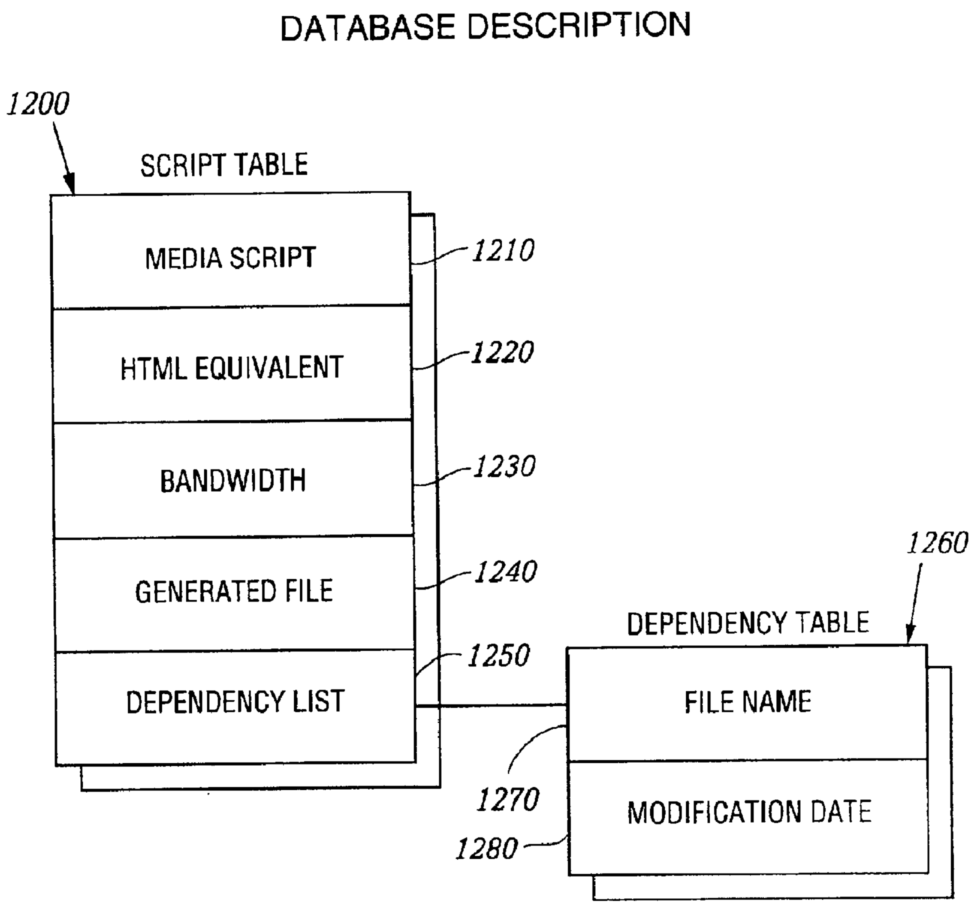

DATABASE DESCRIPTION
1200
SCRIPT TABLE
MEDIA SCRIPT
1210
HTML EQUIVALENT
1220
BANDWIDTH
1230
GENERATED FILE
1240
DEPENDENCY LIST
1250
1260
DEPENDENCY TABLE
FILE NAME
MODIFICATION DATE
1270
1280

FIG. 12

ORIGINAL IMAGES

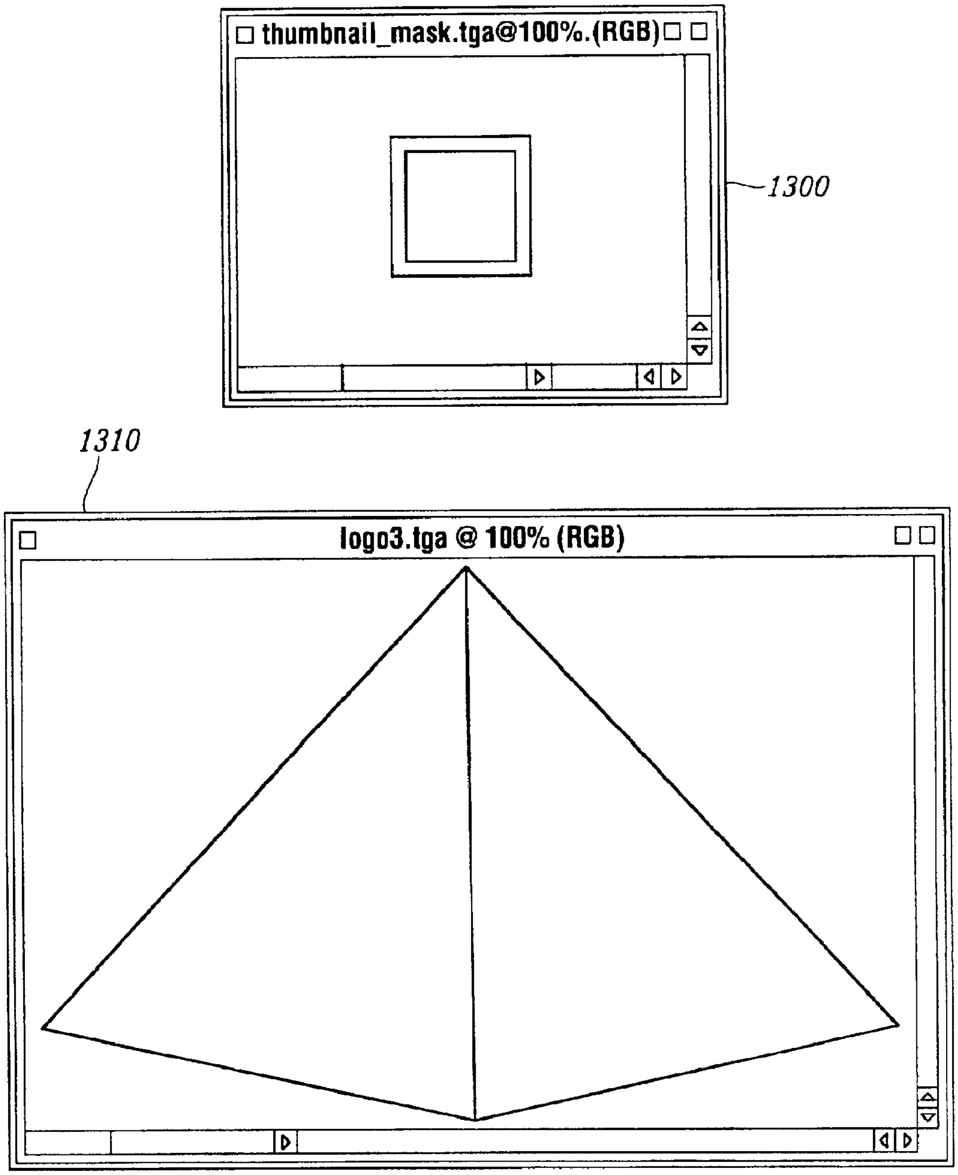
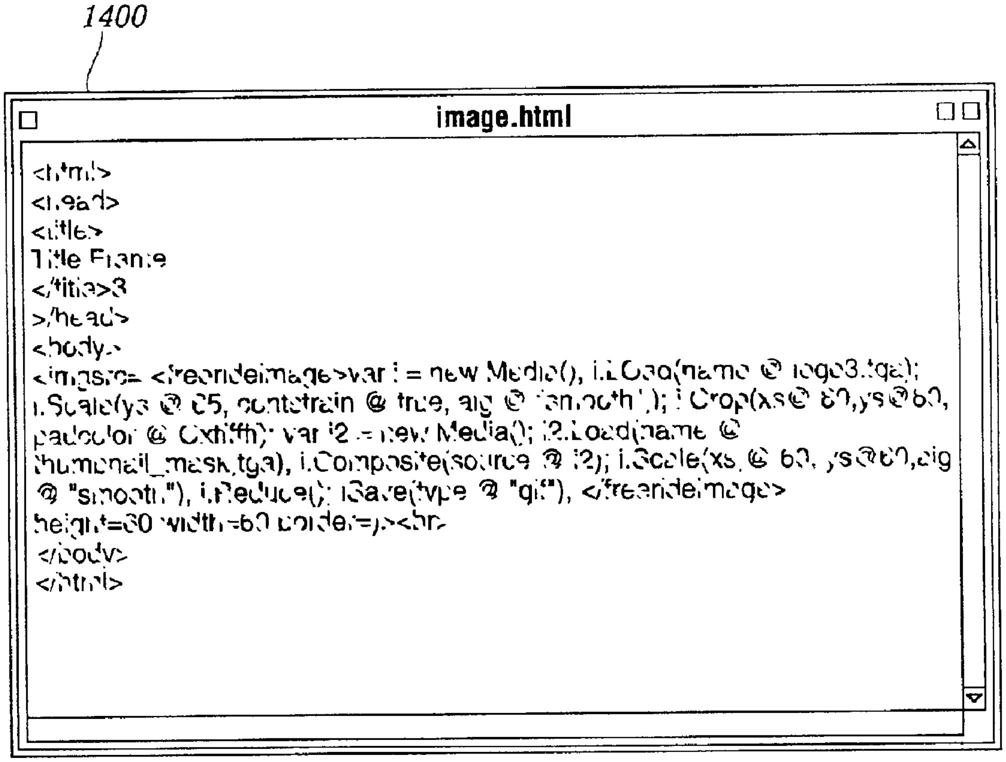


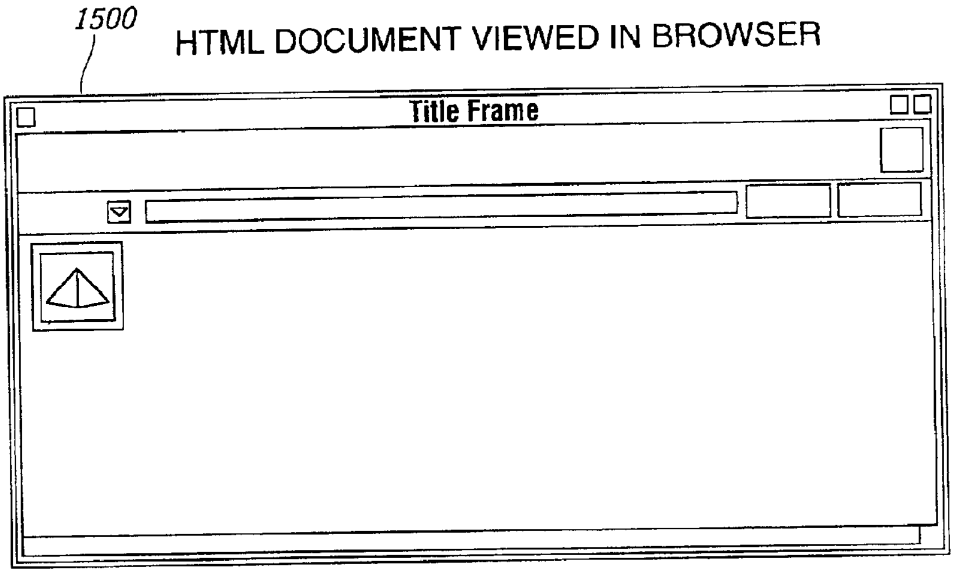
FIG.13

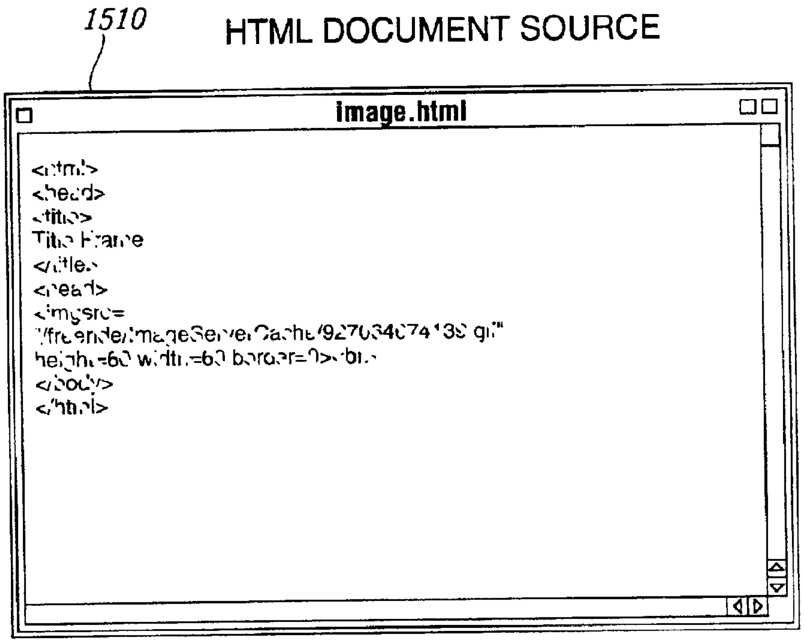
HTML DOCUMENT WITH PROPRIETARY TAG



```
<html>
<head>
<title>
Title Frame
</title>3
>/head>
<body>
<imgsrc= <freerideimage>var i = new Media(), i.Load(name @ logo3.tga);
i.Scale(ys @ 65, contstrain @ true, alg @ "smooth"); i Crop(xs@ 60,ys@60,
padcolor @ Oxffffff); var i2 = new Media(); i2.Load(name @
thumbnail_mask.tga), i.Composite(source @ i2); i.Scale(xs @ 60, ys@60,alg
@ "smooth"), i.Reduce(); iSave(type @ "gif"), </freerideimage>
height=60 width=60 border=0><br>
</body>
</html>
```

FIG.14





```
<html>
<head>
<title>
Title Frame
</title>
<head>
<imgsrc=
"/frontend/ImageServerCache/927034074138.gif"
height=60 width=60 border=0><br>
</body>
</html>
```

FIG.15

GENERATED GIF IMAGE

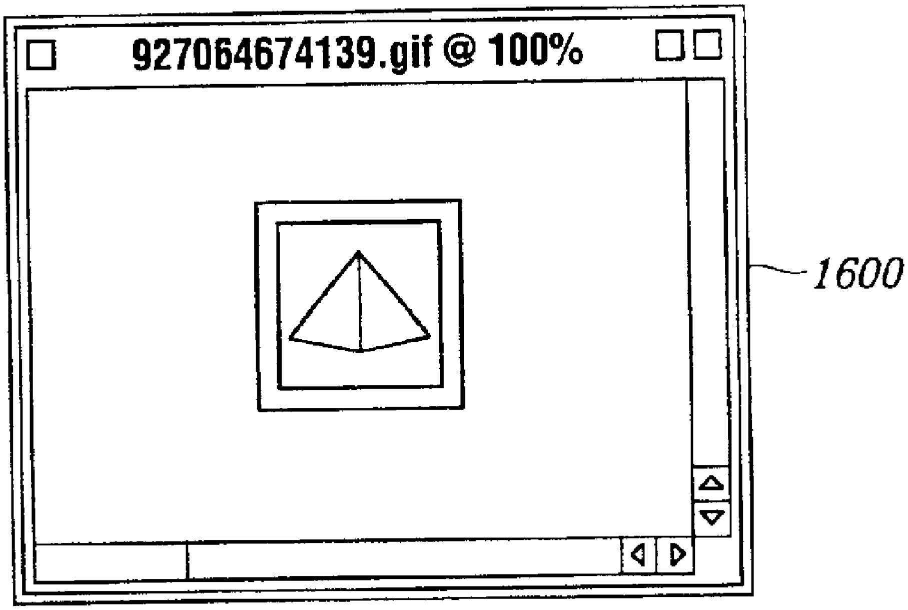


FIG.16

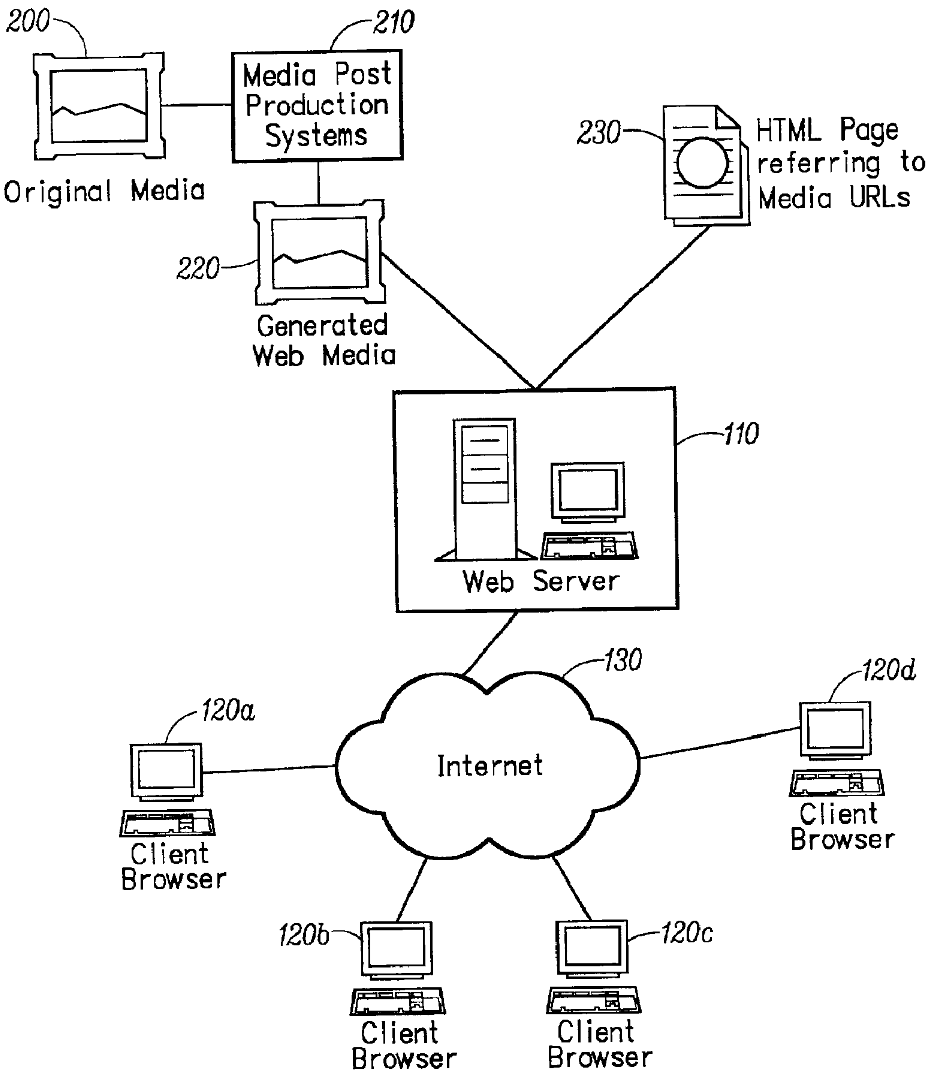

200
Original Media
210
Media Post Production Systems
230
HTML Page referring to Media URLs
220
Generated Web Media
110
Web Server
130
Internet
120a
Client Browser
120d
Client Browser
120b
Client Browser
120c
Client Browser

FIG. 17

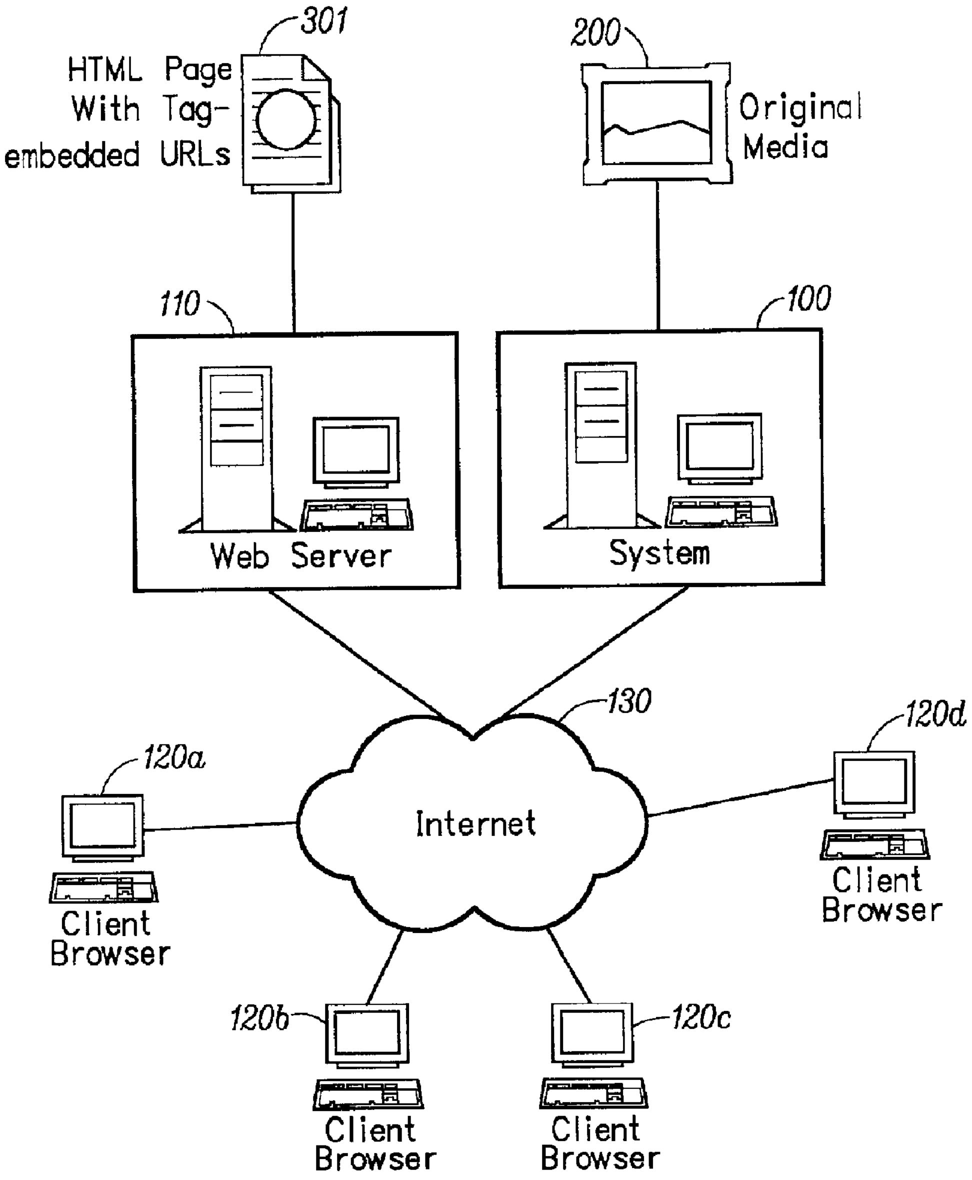

301
HTML Page With Tag-embedded URLs
200
Original Media
110
Web Server
100
System
130
Internet
120a
Client Browser
120b
Client Browser
120c
Client Browser
120d
Client Browser

FIG. 18

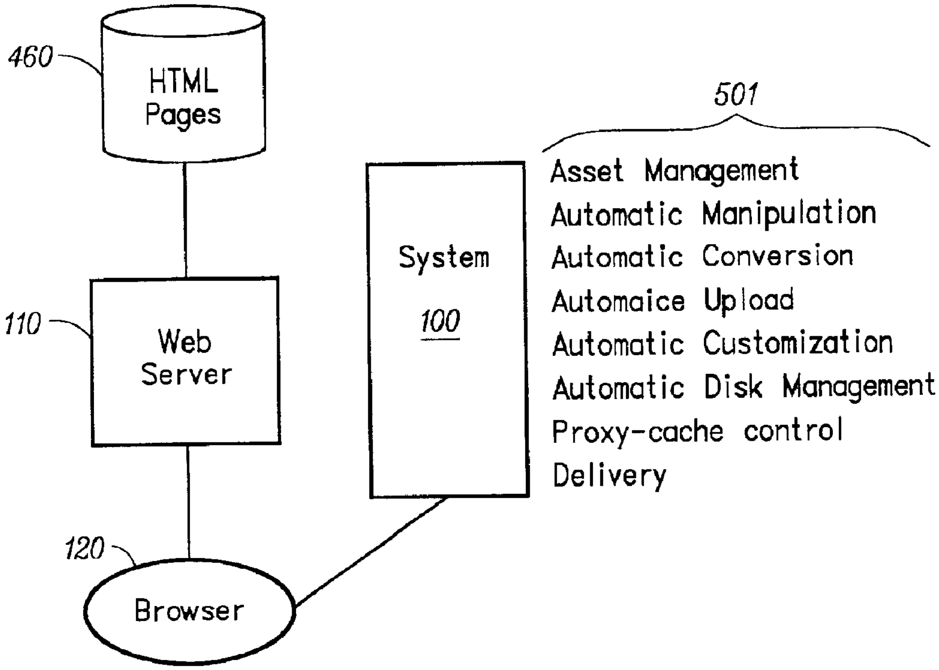

460
HTML Pages
110
Web Server
120
Browser
System
100
501
Asset Management
Automatic Manipulation
Automatic Conversion
Automaice Upload
Automatic Customization
Automatic Disk Management
Proxy-cache control
Delivery

FIG. 19

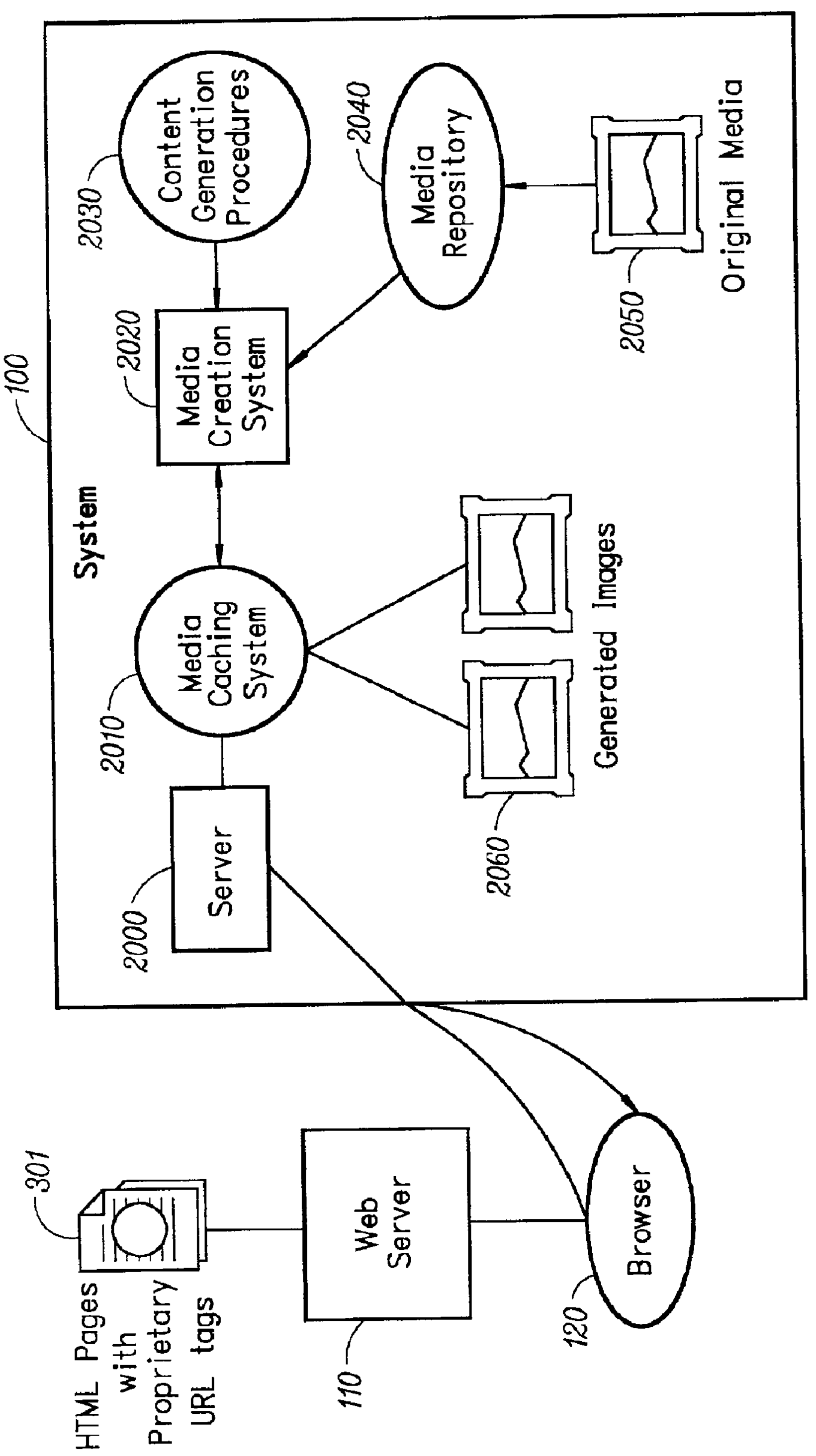

100
System
2030
Content Generation Procedures
2020
Media Creation System
2040
Media Repository
2050
Original Media
2010
Media Caching System
2060
Generated Images
2000
Server
301
HTML Pages with Proprietary URL tags
110
Web Server
120
Browser

FIG. 20

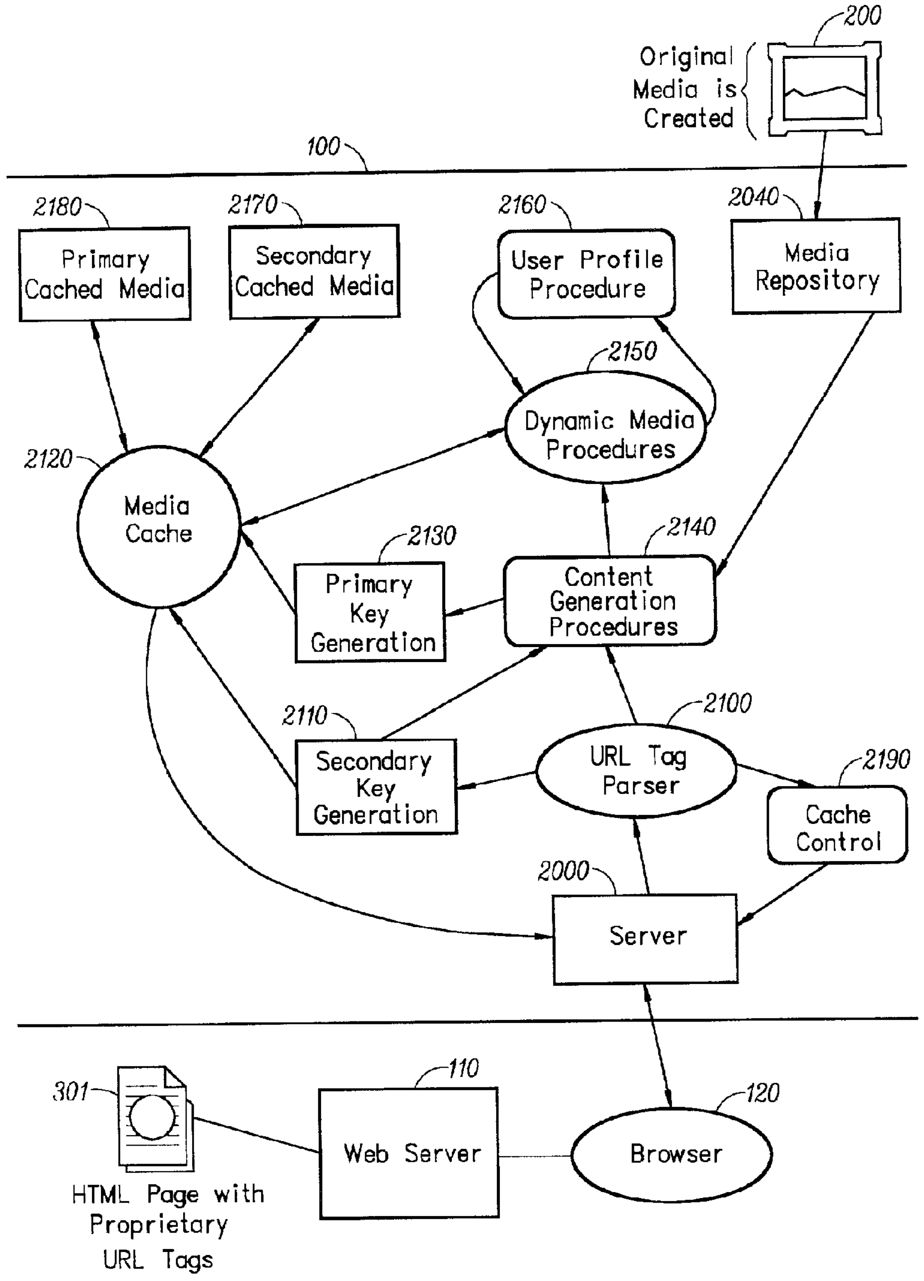

200
Original Media is Created
100
2180
Primary Cached Media
2170
Secondary Cached Media
2160
User Profile Procedure
2040
Media Repository
2150
Dynamic Media Procedures
2120
Media Cache
2130
Primary Key Generation
2140
Content Generation Procedures
2110
Secondary Key Generation
2100
URL Tag Parser
2190
Cache Control
2000
Server
301
HTML Page with Proprietary URL Tags
110
Web Server
120
Browser

FIG. 21

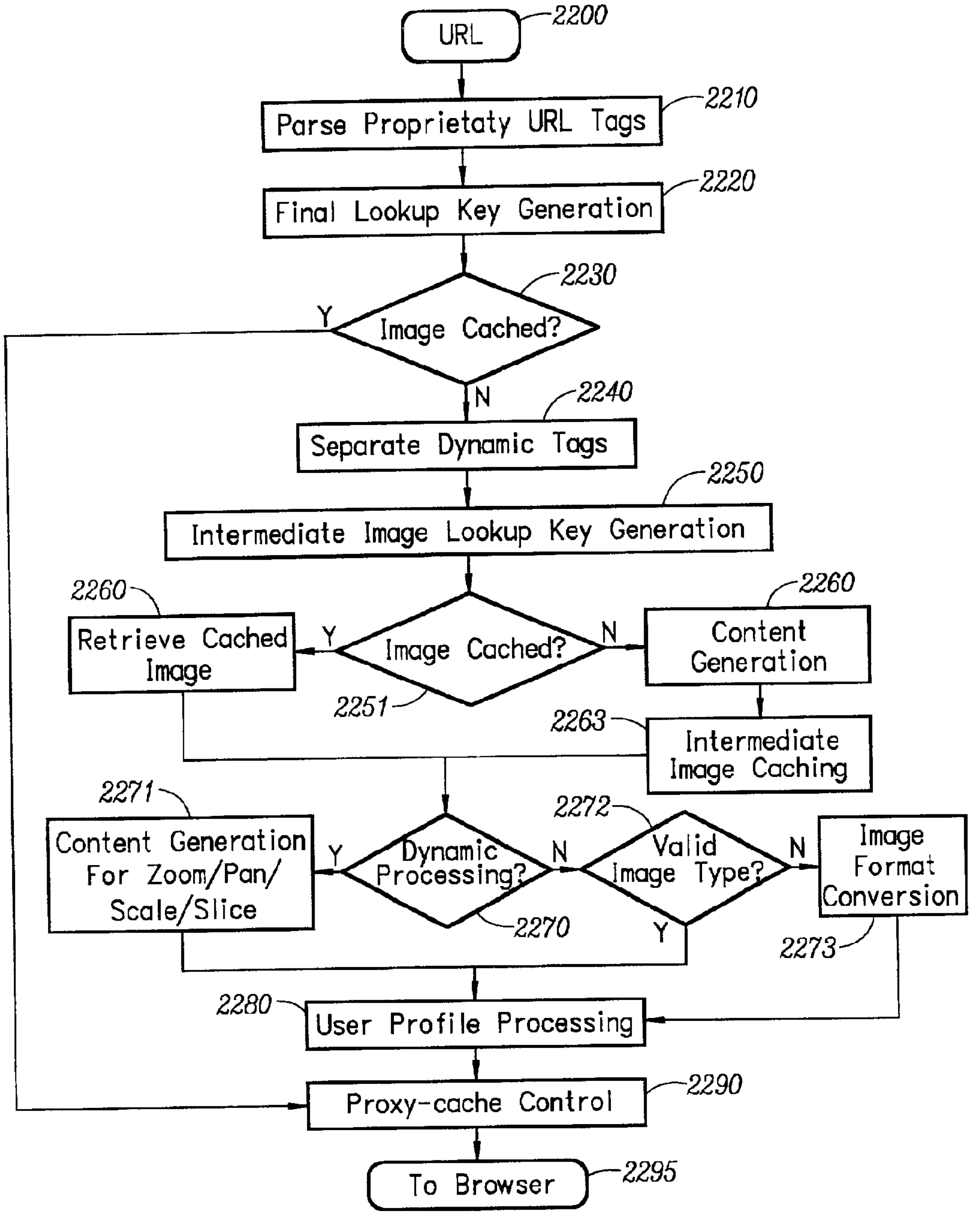

URL
2200
Parse Proprietaty URL Tags
2210
Final Lookup Key Generation
2220
Image Cached?
2230
Y
N
Separate Dynamic Tags
2240
Intermediate Image Lookup Key Generation
2250
2260
Retrieve Cached Image
Y
Image Cached?
N
Content Generation
2260
2251
2263
Intermediate Image Caching
2271
Content Generation For Zoom/Pan/ Scale/Slice
Y
Dynamic Processing?
N
2272
Valid Image Type?
N
Image Format Conversion
2270
Y
2273
2280
User Profile Processing
Proxy-cache Control
2290
To Browser
2295

FIG. 22

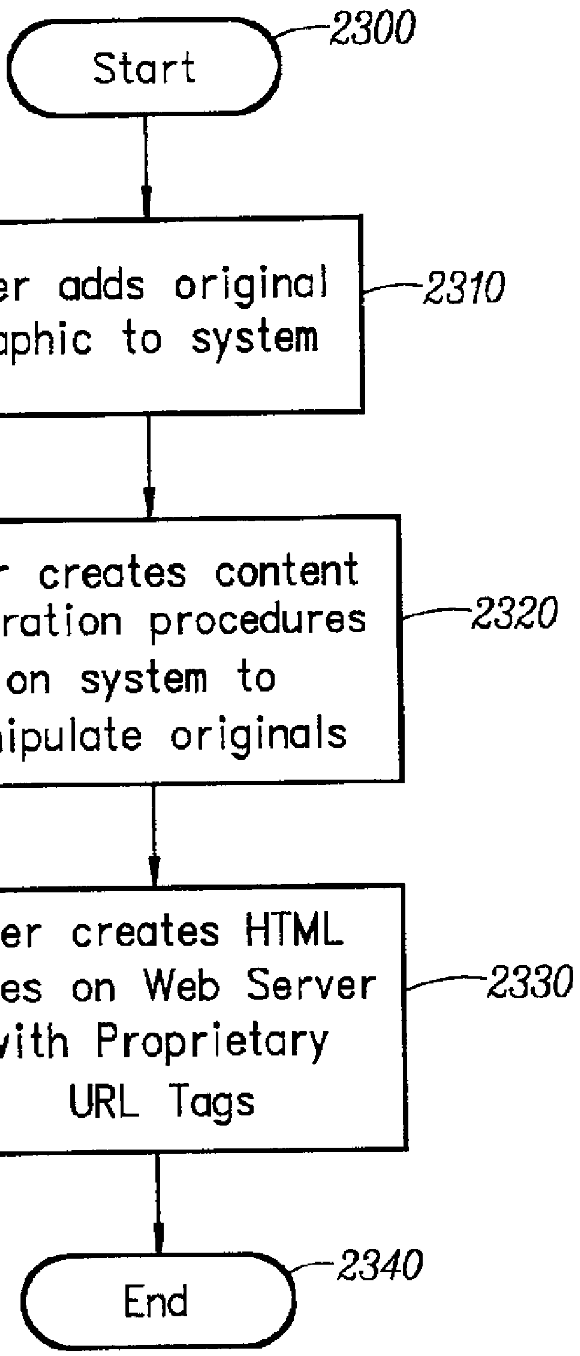

Start
2300
User adds original graphic to system
2310
User creates content generation procedures on system to manipulate originals
2320
User creates HTML pages on Web Server with Proprietary URL Tags
2330
End
2340

FIG. 23

AUTOMATED MEDIA DELIVERY SYSTEM

This APPLICATION is a Continuation-in-part (CIP) of prior application No. 09/425,326 filed Oct. 21, 1999 and is hereby incorporated by reference.

This application claims benefit of Provisional 60/226,043 filed Aug. 16, 2000.

BACKGROUND OF THE INVENTION

1. Technical Field

The invention relates to software systems. More particularly, the invention relates to an Internet server-based software system that provides delivery of automated graphics and other media to Web sites for access by an end user or consumer.

2. Description of the Prior Art

Most Web sites today are primarily handmade. From the guy publishing a simple online technology newsletter from his home, to the Fortune 1000 company's multi-tiered site with hundreds of pages of text, images, and animations, the Web developer and each of his HTML-coding and graphics-producing coworkers toil page by page and image by image. Thousands of established online companies employ hundreds of highly-skilled workers just to produce and maintain their Web sites. After all, the Web is now a major selling vehicle and marketing medium for many of these companies. The Web has even sprouted service industries such as, for example, public companies with multi-billion dollar valuations created just to consult and produce Web sites for others.

Most Web developers who use established WYSIWYG tools in the industry still must produce each page on their Web site one by one. The same rate applies to preparing and placing images, animations, and other visual assets. Each page represents its own set of issues ranging from whether to use GIF, JPEG, or PNG file formats, to finding the optimum bit depth for each image to ensure the fastest downloading through the different browsers of the consumer. The bottlenecked state of the customer's workflow to produce graphics for Web pages can be described as follows:

Current Workflow for Creating Web Graphics

Original Artwork/Asset Creation
  Use third-party point products

Asset Editing
  Scale/reduce/slice

Asset Format Conversion
  JPEG/GIF/PNG

Asset Staging
  Place in Web file system
  Edit HTML

Create/Modify HTML for particular page

Store HTML on Web server

View final pages

Repeat process for each version of each graphic on each page

Estimated time
  Two hours per page times the number of pages

Also, from a user's perspective, the current state of the art is to offer the consumer zooming and panning capabilities so that by clicking on an image the consumer can view more closely or from a different angle. On the horizon are pages with three-dimensional imagery that enable a user to move around a page that can look more like a room than a brochure. While interesting, these features are merely incremental improvements to a consumer's surfing experience.

D. C. A. Bulterman, *Models, Media, and Motion: Using the Web to Support Multimedia Documents,* Proceedings of 1997 International Conference on Multimedia Modeling, Singapore, Nov. 17–20, 1997 discloses "an effort underway by members of industry, research centers and user groups to define a standard document format that can be used in conjunction with time-based transport protocols over the Internet and intranets to support rich multimedia presentations. The paper outlines the goals of the W3C's Synchronized Multimedia working group and presents an initial description of the first version of the proposed multimedia document model and format."

Text and Graphics on UMI's ProQuest Direct: The Best (yet) of both Worlds, Online, vol. 21, no. 2, pp. 73–7, March=14 April 1997 discloses an information system that offers "periodical and newspaper content covering a wide range of business, news, and professional topics . . . letting the user search both text and graphics and build the product to suit. Articles can be retrieved in varying levels of detail: citation, abstracts, full text, and text with graphics. Images come in two flavors: Page Image, a virtual photocopy, and Text+Graphics, in which graphics are stored separately from the text and are manipulable as discrete items . . . . [The system] comes in two versions: Windows and Web."

John Mills Dudley, *Network-Based Classified Information Systems,* AU-A-53031/98 (Aug. 27, 1998) discloses a "system for automatically creating databases containing industry, service, product and subject classification data, contact data, geographic location data (CCG-data) and links to web pages from HTML, XML, or SGML encoded web pages posted on computer networks such as Internets or Intranets . . . . The . . . databases may be searched for references (URLs) to web pages by use of enquiries which reference one or more of the items of the CCG-data. Alternatively, enquiries referencing the CCG-data in the databases may supply contact data without web page references. Data duplication and coordination is reduced by including in the web page CCG-data display controls which are used by web browsers to format for display the same data that is used to automatically update the databases."

Cordell et al, Automatic Data Display Formatting with A Networking Application, U.S. Pat. No. 5,845,084 (Dec. 1, 1998) discloses a placeholder image mechanism. "When a data request is made, the data transfer rate is monitored. When the receive data transfer rate is slow, and the data contains an embedded graphical image of unknown dimensions, a small placeholder image is automatically displayed for the user instead of the actual data. The small placeholder image holds a place on a display device for the data or the embedded graphical image until the data or embedded graphical image is received. When embedded graphical image is received, the placeholder image is removed, and the display device is reformatted to display the embedded graphical image."

Jonathon R. T. Lewis, System For Substituting Tags For Non-Editable Data Sets in Hypertext Documents And Updating Web Files Containing Links Between Data Sets Corresponding To Changes Made To The Tags, U.S. Pat. No. 5,355,472 (Oct. 11, 1994) discloses a "hypertext data processing system wherein data sets participating in the hypertext document may be edited, the data processing system inserting tags into the data sets at locations corresponding to the hypertext links to create a file which is editable by an editor and the data processing system removing the tags, generating a revised data set and updating the link information after the editing process. Its main purpose is to preserve the linking hierarchy that may get lost when the individual data sets get modified."

Wistendahl et al, System for Mapping Hot Spots in Media Content Interactive Digital Media Program, U.S. Pat. No. 5,708,845 (Jan. 13, 1998) discloses a "system for allowing media content to be used in an interactive digital media (IDM) program [that] has Frame Data for the media content and object mapping data (N Data) representing the frame addresses and display location coordinates for objects appearing in the media content. The N Data are maintained separately from the Frame Data for the media content, so that the media content can be kept intact without embedded codes and can be played back on any system. The IDM program has established linkages connecting the objects mapped by the N Data to other functions to be performed in conjunction with display of the media content. Selection of an object appearing in the media content with a pointer results in initiation of the interactive function. A broad base of existing non-interactive media content, such as movies, videos, advertising, and television programming can be converted to interactive digital media use. An authoring system for creating IDM programs has an object outlining tool and an object motion tracking tool for facilitating the generation of N Data. In a data storage disk, the Frame Data and the N Data are stored on separate sectors. In a network system, the object mapping data and IDM program are downloaded to a subscriber terminal and used in conjunction with presentation of the media content."

Rogers et al, Method for Fulfilling Requests of A Web Browser, U.S. Pat. No. 5,701,451 (Dec. 23, 1997) and Lagarde et al, Method for Distributed Task Fulfillment of Web Browser Requests, U.S. Pat. No. 5,710,918 (Jan. 20, 1998) disclose essentially "improvements which achieve a means for accepting Web client requests for information, obtaining data from one or more databases which may be located on multiple platforms at different physical locations on an Internet or on the Internet, processing that data into meaningful information, and presenting that information to the Web client in a text or graphics display at a location specified by the request."

Tyan et al, HTML Generator, European Patent Application No. EP 0843276 (May 20, 1998) discloses "generating an HTML file based on an input bitmap image, and is particularly directed to automatic generation of an HTML file, based on a scanned-in document image, with the HTML file in turn being used to generate a Web page that accurately reproduces the layout of the original input bitmap image."

TrueSpectra has a patent pending for the technology employed in its two products, IrisAccelerate and IrisTransactive. These products are designed for zooming and panning and simple image transformations and conversions, respectively. They support 10 file formats and allow developers to add new file formats via their SDK. They do not require the use of Flashpix for images. However, their documentation points out that performance is dependent on the Flashpix format. The system would be very slow if a non-Flashpix format was used.

TrueSpectra allows the image quality and compression to be set for JPEGs only. The compression setting is set on the server and all images are delivered at the same setting.

TrueSpectra has a simple caching mechanism. Images in the cache can be cleared out automatically at certain times and it does not have any dependency features for image propagation. The Web server needs to be brought down in order to update any original assets.

TrueSpectra does not require plug-ins to operate features such as zooming/panning or compositing. The alternative to plug-ins is using their Javascript or active server page technology. These technologies are used by many Web sites to provide interactivity, but not all Web browsers work correctly with these technologies.

TrueSpectra relies on Flashpix as its native file format and does not support media types such as multi-GIFs and sound formats. Flashpix files are typically larger than most file formats. Access to files is faster for zooming and panning, but appears to be quite slow.

The key to IrisTransactive is the compositing subsystem. It requires three things to build a shopping solution using image composition.

1) The original images must be created. It is suggested that the image be converted to Flashpix for better performance.

2) All of the individual images must be described in XML using the image composer program. The program allows the editor to specify anchor points, layer attributes, and layer names. The resulting file is between 5 k and 50 k.

3) The Web designer must place HTML referring to the XML in the Web site. By specifying parameters to the XML, the Web designer can turn on or off layers.

The herein above process for compositing images enables Web designers to create shopping sites. However, a lot of overhead is the result. The XML documents add 5 k–50 k to a Web site. The compositing commands that are embedded in the HTML are difficult to understand. And, because the compositing feature requires several steps to implement, it is not suitable for every image on a Web site. The process seems to be designed for the specific purpose of shopping.

MediaBin(TM) is limited to activities behind the firewall automating only the "post-creative busywork." In addition, MediaBin requires the use of an application server to function through a web interface. Thus images may not be directly added to any existing web page.

Macromedia's Generator operates by embedding variables in their proprietary Flash format. Therefore the actual imaging operations are somewhat limited and cannot be controlled directly from a web page request.

MGI Software sells point solutions that require end-users to download a viewer to process a proprietary image format.

PictureIQ offers a server-side image-processing appliance that provides a limited set of Photoshop functionalities. This appliance runs on the web-page server, processes information embedded in the web page, and rewrites the web page with image data.

The disclosed prior art fail to provide systems and methodologies that result in a quantum leap in the speed with which they can modify and add images, video, and sound to sites, in the volume of data they can publish internally and externally, and in the quality of the output. The development of such an automated media delivery system would constitute a major technological advance.

It would be advantageous to empower an end user with flexibility and control by providing interactive page capabilities.

It would be advantageous from an end user's perspective to generate Web pages that contain active graphics. For example, clicking on a Corvette image will cause a simple menu to pop up suggesting alternative colors and sizes in which to see the car. Clicking on portions of the image, such as a fender, can call up a close-in view of the fender.

It would be advantageous to provide an automated graphics delivery system that becomes part of the Web site infrastructure and operates as part of the Web page transaction and that thereby provides a less expensive and less time-consuming process.

It would be advantageous to provide a system for automated processing and delivery of media (images, video, and sound) to a Web server whereby it eliminates the laborious post-production and conversion work that must be done before a media asset can be delivered on a Web server.

It would be advantageous to create a dynamic Web site, wherein images are generated on demand from original assets, wherein only the original assets need to be updated, and wherein updated changes propagate throughout the site.

It would be advantageous to provide a system that generates media based on current Web server traffic thereby optimizing throughput of the media through the Web server.

It would be advantageous to provide a system that generates media that is optimized for the Web client, wherein client connection speed determines optimum quality and file size.

It would be advantageous to provide a system that generates media, whereby the media is automatically uploaded.

It would be advantageous to provide a system that automatically caches generated media so identical requests can be handled without regeneration of images.

It would be advantageous to provide a system that resides behind the Web server, thereby eliminating security issues.

It would be advantageous to provide a system wherein the client browser does not require a plug-in.

It would be advantageous to provide a system wherein the system does not require any changes to a Web server.

It would be advantageous to provide a system wherein the system manages the Web server media cache.

It would be advantageous to provide a system wherein the Web media is generated only if requested by a client browser.

It would be advantageous for a system to reduce the need for a Web author to create different versions of a Web site, the system automatically handling image content.

It would be advantageous to provide dynamic imaging capabilities, have a more complete set of image processing functionality, and be controlled directly through an image URL.

It would be advantageous to provide an end-to-end solution requiring only a standard browser that is completely controllable using the proprietary tags contained within a simple image link in the web page.

It would be advantageous to run an image application as a separate server controlled directly by single image requests to that server, such that any web server, even one that is only sending static HTML can access imaging features.

SUMMARY OF THE INVENTION

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, generated media is cached such that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs

for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a schematic diagram showing the placement of the system within a current Web infrastructure according to the invention;

FIG. **2** is a schematic diagram showing how a typical Web site delivers an HTML document and its graphics to a Web browser according to the prior art;

FIG. **3** is a schematic diagram showing delivery of an HTML document and media to a Web browser according to the invention;

FIG. **4** is a schematic diagram showing the components involved in Web site administration according to the prior art;

FIG. **5** is a schematic diagram showing the components of the system involved in Web site administration according to the invention;

FIG. **6** is a simple overview showing the components of the system according to the invention;

FIG. **7** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to the invention;

FIG. **8** is a flow chart showing an authoring process according to the invention;

FIG. **9** is a flow chart showing an HTML parsing process according to the invention;

FIG. **10** is a flow chart showing a media creation process according to the invention;

FIG. **11** is a screen shot showing an administration tool according to the invention;

FIG. **12** displays a structure of a database record used for the system according to the invention;

FIG. **13** shows original media to be processed according to the invention;

FIG. **14** shows a portion on an HTML document with a proprietary tag according to the invention;

FIG. **15** shows an HTML document and an HTML document source according to the invention;

FIG. **16** shows a generated GIF image according to the invention;

FIG. **17** is a schematic diagram of an image system within a typical Web infrastructure according to the invention;

FIG. **18** is a schematic diagram showing delivery of an HTML document and original media according to the invention;

FIG. **19** is a schematic diagram showing components of Web site administration according to a preferred embodiment of the invention;

FIG. **20** is a simple overview showing components of the image system according to a preferred embodiment of the invention;

FIG. **21** is a schematic diagram showing process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention;

FIG. **22** shows a flowchart of a content generation procedure according to a preferred embodiment of the invention; and

FIG. **23** is a flow chart showing an authoring process according to a preferred embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided.

The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, the generated media is cached so that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

A detailed description of such automatic media delivery system operating in parallel with existing Web site infrastructure is found below in the section under the heading as such.

FIG. **1** is a schematic diagram showing the placement of the system within a current Web infrastructure according to a preferred embodiment of the invention. The system **100** is attached to a Web server **110**, which is connected to multiple client browsers **120**(*a–d*) via the Internet **130**.

FIG. **2** is a schematic diagram showing how a typical Web site delivers an HTML document and its graphics to a Web browser according to the prior art. An original media **200** is passed to post-production systems **210**, wherein the media **200** is manipulated by hand and prepared for the Web. The result is a Web media **220**. The Web media **220** and an associated HTML document **230** referring to the media **220** by media tags are input to a Web server **110** for a Web browser **120** to view via the Internet **130**.

FIG. **3** is a schematic diagram showing delivery of an HTML document and media to a Web browser according to a preferred embodiment of the invention. An original media **200** and an HTML document embedded with proprietary media tags **300** are input into the system **100**. The system **100** generates a Web-safe media **220** and a modified HTML document **230** that refers to the Web media, and automatically loads them onto the Web server **110** for view by a Web browser **120** via the Internet **160**.

FIG. **4** is a schematic diagram showing components involved in Web site administration according to the prior art. Original media assets **400** are original images, video, or sound that have not been prepared for the Web. Web sites usually need to manage the placement of media on the network for easy retrieval by Web designers. Post-production systems **410** vary from Web site to Web site. Post-production systems **410** are usually custom procedures that Web designers use to convert an original media, such as an image, to one that can be displayed on the Web. Post-production systems **410** also upload finished images to Web image systems. Web images **420** are Web versions of the original images. Web images **420** are ready for retrieval by the Web server **110** to be delivered to a Web browser **120**. Any image to be modified or updated must pass through the herein above three components before it can be delivered to the Web browser **120**. HTML pages **460** have references to Web images **420**.

FIG. **5** is a schematic diagram showing the components involved in Web site administration according to a preferred embodiment of the invention. Web site administration is simplified using the claimed invention. Asset management, automatic image manipulation, automatic image conversion, automatic image upload, and automatic disk management **500** are provided by the claimed invention.

FIG. **6** is a simple overview showing the components of the system according to a preferred embodiment of the invention. HTML with proprietary tags **300** is the original HTML document that is embedded with proprietary tags which describe how the images are to be manipulated for the Web. Java servlet engine **600** is a third-party product that allows the system **100** to interface with the Web server **110** and execute Java servlet code. The Web server **110** is third-party software that delivers Web pages to a Browser **120**. The Browser **120** views Web pages that are sent from the Web server **110**. Modified HTML with system created images **230** are a final result of the system. Modified HTML **230** is a standard HTML document without proprietary embedded tags and with standard Web graphics.

The System.

A preferred embodiment of the system **100** is provided.

HTML parsing subsystem **610** parses through an HTML document and searches for proprietary tags. If it finds a proprietary tag it hands it to a media caching subsystem **620** for further processing. The media caching subsystem **620** returns a standard HTML tag. The HTML parsing subsystem **610** then replaces the proprietary tag it found with the returned tag. The parsing subsystem **610** then continues searching for a next proprietary tag, repeating the process herein above. The process is finished when no more proprietary tags can be found.

The media caching subsystem **620** determines if an image has been created for the requested proprietary tag. If the image has already been created and the files that built that image have not been modified, the media caching subsystem **620** returns an HTML tag that refers to a previously-generated image. If the image has not been created, the media caching subsystem **620** hands the HTML tag to a media creation subsystem **630**. The media creation subsystem **630** returns an image to the media caching subsystem **620**. The media caching subsystem **620** adds the created image and the HTML tag to a media cache database **640**.

The media cache database **640** contains references to the created images **645**. In a preferred embodiment, the references are the script used to create the image, the names of the images used to create the image, the dates of those files, and the HTML that represents the created image. The media caching subsystem **620** performs lookups in this database to determine if the image has been created. If the image has not been created the media caching subsystem **620** calls upon the media creation subsystem **630** to create the image and then store the results in the media cache database **640**.

The media creation subsystem **630** takes a proprietary tag from the media caching subsystem **620** and generates an image. The image is generated by deciphering the tag and handing it to the media processing engine **650**. After the image is created, the media creation subsystem returns the name of the newly created image to the media caching subsystem **620**.

The media processing engine **650** interprets the proprietary tag and generates the image. The media processing engine **650** looks up images in a media repository to obtain the location of the original file.

The media repository **660** contains original images **665** used in the system **100**.

FIG. **7** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention. An original media **200** is created. The media **200** is placed into the system **100** in the media repository **660**. Similarly, an HTML document with proprietary tags **300** is created and placed on a Web server **110**. A user requests a Web page from a Web browser **120**. The Web server **110** passes the requested page to an HTML parser **610**. The HTML parser **610** parses HTML looking for media tags. The parser **610** looks up media tags in a media tags database **640**. If the media tag is found, then the system **100** produces a modified HTML document **230**. Otherwise, the media creation subsystem **630** uses the media tag to generate a Web media **220**. The generated Web media **220** is placed in a media cache subsystem **620**. The proprietary media tag is converted by a converter **700** to a standard HTML tag that refers to the generated media **220** in cache. The media tag and the HTML equivalent are stored in the media tags database **640**. Media tags are replaced by standard HTML equivalent to provide a modified HTML document **230**. The modified HTML document **230** is delivered to the Web server **110**. The Web server **100** delivers the modified HTML document **230** to the browser **120** via the Internet for a user to view.

FIG. **8** is a flow chart showing an authoring process according to a preferred embodiment of the invention. The process starts (**800**) when a user adds an original graphic to the system (**810**). The user then creates an HTML document that contains proprietary media tags (**820**). The user then places the HTML document on a Web server (**830**) and ends the authoring process (**840**).

FIG. **9** is a flow chart showing an HTML parsing process according to a preferred embodiment of the invention. The process starts (**900**) when a consumer requests a Web page (**910**). A Web server hands the request of the Web page to the system (**920**). The system parses the Web page (**930**). The system looks for a media tag (**940**). If found, the system retrieves the HTML equivalent of the media tag (**950**) and replaces the media tag with the HTML equivalent tag (**960**). The system continues parsing the Web page for tags (**970**) by returning to step (**940**). When no more tags are found, the system delivers the modified Web page to the Web server (**980**) and therein ends the process (**990**).

FIG. **10** is a flow chart showing a media creation process according to a preferred embodiment of the invention. The process starts (**1000**) when the system requests an HTML equivalent to a proprietary media tag (**1010**). The Media tag is combined with bandwidth information (**1020**). The subsystem checks if the media tag already exists in the media tag database (**1030**). If It does, the subsystem checks if any of the original assets used to create the media have been changed (**1040**). If not, then the subsystem retrieves the HTML equivalent tag from the database (**1050**) and returns the HTML equivalent tag to the requesting system (**1060**). If any of the original assets used to create the media have been changed (**1040**), then the subsystem removes the media tag entry from the media database (**1070**) and creates the media using the media tag (**1080**). The subsystem then stores the media in a media cache (**1090**). The subsystem generates the HTML referring to the generated media (**1100**) and places the media tag and the HTML equivalent in the media tag database (**1110**). The HTML equivalent is returned to the requesting system (**1060**) and the process stops (**1120**).

The differences between using HTML and the proprietary tags disclosed herein are noted. HTML allows Web designers to create Web page layouts. HTML offers some control of the images. HTML allows the Web designer to set the height and width of an image. However, all of the other image operations disclosed herein are supported by the claimed invention and are not supported by HTML.

Table A herein below provides the claimed proprietary tags according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

TABLE A

Tags
Generate image
<freerideimage> mediascript </freerideimage>
Generate a standard Web image.
Generate thumbnail image linked to full image
<freerideimagethumbnail> mediascript <xs=size ys=size /freerideimagethumbnail>
Generate a thumbnail of specified size and link it to the full size version.
Generate zoom and pan image
<freerideimagezoom> mediascript </freerideimagezoom>
Generate a zoomable/panable image.
Security
<freerideimagesecure> </freerideimagesecure>
Specifies that all images found between these tags are secured images and the system will determine access before generating.

Table B herein below provides the claimed script commands according to a preferred embodiment of the invention. Additional commands may be added as needed.

TABLE B

Media processing script commands
Add Noise
Noise_AddNoise([amount=<value 1..999>] [gaussian] [grayscale])
This command adds noise to the image.
Adjust HSB

TABLE B-continued

Media processing script commands

AdjustHsb([hue @ <value ±255>] [saturation @ <value ±255>] [brightness @ <value ±255>])
This command allows the HSB of an image to be altered. This can be applied to images of all supported bit-depths.
Adjust RGB
AdjustRgb([brightness @ <value ±255>] [contrast @ <value ±255>] [red @ <value ±255>] [green @ <value ±255>] [blue @ <value ±255>] [noclip @ <true, false>] [invert @ <true, false>])
This command allows the contrast, brightness, and color balance of an image to be altered.
Blur
Blur(radius @ <value 0..30>)
This command applies a simple blur filter on the image.
Blur Convolve
Blur__Blur( )
This command commands perform a simple 3×3 convolution for blurring.
Blur Convolve More
Blur__MoreBlur( )
This command commands perform a stronger 3×3 convolution for blurring.
Blur Gaussian
Blur__GaussianBlur([radius=<value 0.1..250>])
This command applies a Gaussian blur to the image.
Blur Motion
Blur__MotionBlur([distance=<value 1..250>] [angle=<degrees>])
This command applies motion blurring to the image using the specified distance and angle.
Brush Composite
Composite(source @ {<User-Defined Media Object name>} [x @ <pixel>] [y @ <pixel>] [onto] [opacity @ <value 0..255] > [color @ <color in hexadecimal>] [colorize @ <true, false>] [saturation @ <value 0..255>])
This command composites the specified "brush" (foreground) image onto the current "target" (background) image.
Colorize
Colorize(color @ <color in hexadecimal> [saturation @ <value 0..255>])
This command changes the hue of the pixels in the image to the specified color.
Convert
Convert(rtype @ <bit-depth> {dither @ <value 0..10>])
This command converts the image to the specified type/bit-depth.
Convolve
Convolve(Filter @ <filtername>)
This command applies a basic convolution filter to the image. In a user interface driven system, the filters could be stored in files and edited/created by the user.
Crop/Resize Canvas
Crop([xs @ {<pixels>, <percentage + "%">}] [ys @ {<pixels>, <percentage + "%">}] [xo @ <left pixel>]
[yo @ <top pixel>] [padcolor @ <color in hexadecimal>] [padindex @ <value 0..255>])
This command crops the media to a specified size.
Discard
Discard( )
This command removes the designated Media Object from memory.
Drop Shadow
DropShadow([dx @ <pixels>] [dy @ <pixels>] [color @ <color in hexadecimal>] [opacity @ <value 0..255>] [blur @ <value 0..30> [enlarge @ <true, false>])
This command adds a drop shadow to the image based on its alpha channel.
Equal
Equal(source @ {<User-Defined Media Object name>})
This command compares the current media with the one specified. If the media are different in any way, an error value is returned.
Equalize
Equalize([brightness @ <−1, 0..20>] [saturation @ <−1, 0..20>])
This command equalizes the relevant components of the media. Equalization takes the used range of a component and expands it to fill the available range.
Export Channel
ExportGun(Channel @ <channelname>)
This command exports a single channel of the source as a grayscale image.
Find Edges
Stylize__FindEdges([threshold=<value 0..255>] [grayscale] [mono] [invert])
This command finds the edges of the image based on the specified threshold value.
Fix Alpha
FixAlpha( )
This command adjusts the RGB components of an image relative to its alpha channel.
Flip
Flip(<horizontal, vertical> @ <true, false>)
This command flips the media vertically or horizontally.
Frame Add
FrameAdd(Source @ <filename>)
This command adds the given frame(s) to the specified Media Object.
Glow/Halo
Glow(Size @ <value 0..30> [halo @ <value 0..size>] [color @ <color in hexadecimal>] [opacity @ <value 0..255>] [blur @ <value 0..30>] [enlarge @ <true, false>])
This command produces a glow or halo around the image based on the image's alpha.

TABLE B-continued

Media processing script commands
High Pass Other__HighPass([radius=<value 0.1..250>]) This command replaces each pixel with the difference between the original pixel and a Gaussian blurred version of the image.
Import Channel ImportGun(channel @ <channel name> source @ {<User-Defined Media Object name>} [rtype @ <bit-depth>]) This command imports the specified source image (treated as a grayscale) and replaces the selected channel in the original.
Load Load(Name @ <filename> [type @ <typename>] [transform @ <true, false>]) This command loads a media from the specified file.
Maximum Other__Maximum([radius=<value 1..10>]) This command scans the area specified by the radius surrounding each pixel, and then replaces the pixel with the brightest pixel found.
Minimum Other__Minimum([radius=<value 1..10>]) This command scans the area specified by the radius surrounding each pixel, and then replaces the pixel with the darkest pixel found.
Normalize Normalize([clip @ <value 0.20>]) This command expands the volume of the sample to the maximum possible.
Pixellate Mosaic Pixellate__Mosaic([size=<value 2..64>]) This command converts the image to squares of the specified size, where each square contains the average color for that part of the image.
Pixellate Fragment Pixellate__Fragment([radius=<value 1..16>]) This command produces four copies of the image displaced in each direction (up, down, left, right) by the specified radius distance and then averages them together.
Quad Warp QuadWarp([tlx=<position>] [tly=<position>] [trx=<position>] [try=<position>] [blx=<position>] [bly=<position>] [brx=<position>] [bry=<position>] [smooth]) This command takes the corners of the source image and moves them to the specified locations, producing a warped effect on the image.
Reduce to Palette Reduce([colors @ <num colors>] [netscape @ <true, false>] [b&w @ <true, false>] [dither @ <value 0..10>] [dithertop @ <value 0..10>] [notbackcolor] [pad @ <true, false>]) This command applies a specified or generated palette to the image.
Rotate Rotate(Angle @ <value 0..359> [smooth @ <true, false>] [enlarge @ <true, false>] [xs @ <pixels>] [ys @ <pixels>]) This command rotates the media by the specified angle in degrees.
Rotate 3D Rotate3d([anglex @ <angle ±89>] [angley @ <angle ±89>] [distance @ <value>]) This command rotates the image in 3D about either the x-axis or y-axis.
Save Save([type @ <image-type>]) This command saves a media to the specified file.
Scale Scale([xs @ {<pixels>, <percentage + “%”>}] [ys @ {<pixels>, <percentage + “%”>}] [constrain @ <true, false>] [alg @ {“fast”, “smooth”, “outline”)] [x1 @ <pixels>] [y1 @ <pixels>] [x2 @ <pixels>] [y2 @ <pixels>]) This command scales the image to the specified size.
Select Selection([source @ <User-Defined media Object>}] [remove @ <true, false>] [invert @ <true, false>] [backcolor] [color=<color>] [index=<value>] [opacity @ <value 0..255>]) This command manages the selected region for the current Media Object.
Set Color SetColor([backcolor @ <color in hexadecimal>] [forecolor @ <color in hexadecimal>] [backindex @ <value 0..255>] [foreindex @ <value 0..255>] [transparency @ (“on”,“off”)]) This command allows the background color, foreground color, and transparency state of an image to be set.
Set Resolution SetResolution([dpi @ <value>] [xdpi @ <value>] [ydpi @ <value>]) This command changes the DPI of the image in memory.
Sharpen Sharpen__Sharpen( ) This command sharpens the image by enhancing the high-frequency component of the image.
Sharpen More Sharpen__SharpenMore( ) This command sharpens the image by enhancing the high-frequency component of the image, but is stronger than the standard sharpening.

TABLE B-continued

Media processing script commands
Stylize Diffuse
Stylize_Diffuse([radius=<value 0..>] [lighten] [darken])
This command diffuses the image by randomizing the pixels within a given pixel radius.
Stylize Embose
Stylize_Emboss([height=<value 1..10>] [angle=<degrees>] [amount=<percentage 1..500>])
This command converts the image to an embossed version.
Text Drawing
DrawText(Text @ <string> Font @ <font file> [size @ <value>] [color @ <color in hexadecimal>] [smooth @ <true, false>] [<left, right, top, bottom> @ <true, false>] [x @ <pixel>] [y @ <pixel>] [wrap @ <pixel-width>] [justify @ {left,center,right}] [angle @ <angle>])
This command composites the specified text string onto the image.
Text Making
MakeText(text @ <string> font @ <font file> [path @ <path to font directory>] [size @ <value 1..4095>] [color @ <color in hexadecimal>] [smooth @ <true, false>] [wrap @ <pixel-width>] [justify @ {left,center,right}] [angle @ <angle>])
This command creates a new image that includes only the specified text.
Trace Contour
Stylize_TraceContour([level=<value 0..255>] [upper] [invert])
This command traces the contour of the image at the specified level (for each gun).
Unsharpen Mask
Sharpen_UnsharpMask([amount=<percentage 1..500>] [radius=<value 0.1..250>] [threshold=<value 0..255>])
This command enhances the edges and detail of an image by exaggerating differences between the image and a gaussian blurred version of the same image.
Zoom
Zoom([xs @ <pixels>] [ys @ <pixels>] [scale @ <value>] [x @ <left pixel>] [y @ <top pixel>])
This command zooms in on a specified portion of the media and fits it to the specified size. This constitutes a crop followed by a scale.

Table C herein below provides a list of features provided by a preferred embodiment of the invention. It is noted that the list of features included in Table C is by no means complete. In other embodiments, the list of features is expanded or reduced as needed.

TABLE C

System Feature List
Reads and writes various file formats:
BMP, GIF, JPG, PNG, TIF, PICT, TGA, PSD, FPX;
Supports many image processing operations;
Dynamically creates Web images from original assets;
Dynamically creates thumbnail images;
Dynamically creates images that can be panned and zoomed without browser plug-ins or special file formats;
Automatically propagates changes of original assets throughout a Web site;
Uses an intelligent caching mechanism:
Clean up image cache on demand;
Eliminates orphaned image files; and
Optimizes Web server cache by providing most recent images;
Renders TrueType fonts on the server instead of browser;
Uses intelligent scaling of line drawings;
Allows Web designers to manipulate images with proprietary tags;
Preserves original image assets;
Optimizes Web server traffic by adjusting the bandwidth of graphics;
Optimizes images for client connection speed;
Allows clients to specify the quality of images on a Web site; and
Allows Web designers to dynamically create images by manipulating proprietary tags in their applications (server or client side).

FIG. **11** is a screen shot showing an administration tool according to a preferred embodiment of the invention. Specifically, FIG. **11** shows an administration page that contains cached images of generated scripts. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **12** displays a structure of a database record used for the system according to a preferred embodiment of the invention. A Script Table **1200** has 5 columns, Media Script **1210**, HTML Equivalent **1220**, Bandwidth **1230**, Generated File **1240**, and Dependency List **1250**. A Dependency Table **1260** has two columns, File Name **1270** and Modification Date **1280**.

Snowboard Store Example.

Background.

The snowboard store highlights several features of the claimed system. The snowboard store is an imaginary store that allows a user to configure his or her snowboard. The store consists of five logos, five board colors, and four boards. The consumer clicks on the buttons to change the snowboard represented in the middle of the screen. When the consumer has configured the snowboard they the snowboard can be purchased by selecting a buy button.

Prior Art Method.

To create the snowboard site today, the Web designer must render all possible combinations of the board. The number of combinations is five logos×five board colors×four boards=100. The designer also must render all the buttons. The creation process is very tedious and involves a lot of production work. Typically, most Web sites do not even attempt such an endeavor. Also, other issues must be addressed, such as, for example, updating the Web site and scripting. For example, updating a single logo involves updating a minimum of 20 images.

The prior art method sustains a graphic intensive site that requires management of at least 100 images. Updates to the Web site are time-consuming and prone to human error.

The Claimed Method.

A preferred embodiment of the method scripts the image creation process in HTML to create a dynamic Web site. There is no need to create over 100 images. The claimed system generates images on demand. The Web site only needs to create original assets. The scripting process involves writing the proprietary scripts. In the current example herein, scripting buttons is very simple. Once one button is created, simply copy and paste the HTML to create another button or many buttons. Only the name of the image to be overlaid on the button must be changed. The Webmaster then creates a simple program that reads what object a user has clicked on and generates a proprietary tag. The tag is then sent to the claimed system to generate a center image.

The claimed method allows the creation of all 100 combinations automatically. When the Web site receives an updated image, only the original image needs to be updated. Any change to the original image automatically propagates throughout the system. The Web site is easier to manage. Testing of the Web site is easier because there is no need to test all 100 combinations. A small subset of combinations will guarantee adequate coverage.

Processing of an Image Tag Example (FIGS. **13**–**16**).

FIG. **13** shows two original images **1300** and **1310** to be processed according to a preferred embodiment of the invention.

FIG. **14** shows a portion on an HTML document with a proprietary tag **1400**, <freerideimage></freerideimage> according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **15** shows an HTML document **1500** as viewed in a browser and an HTML document source **1510**, according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **16** shows a generated GIF image **1600** according to a preferred embodiment of the invention.

Automatic Media Delivery System Operating in Parallel with Existing Web Site Infrastructure.

It should be noted that the words, media, graphics, and images are used herein interchangeably.

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, the generated media is cached so that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

A preferred embodiment of the invention is described with reference to FIG. **17**. FIG. **17** is a schematic diagram of an image system within a typical Web infrastructure according to the invention. The image system **100** is placed in parallel to an existing Web server **110**. The image system **100** may be on-site or off-site to the Web infrastructure. Multiple client browsers **120***a*–**120***d* communicate with both the Web server **110** and the image server **100** via the Internet **130**.

The delivery of an HTML document and media according to a preferred embodiment of the invention is described with reference to FIG. **18**. Resource locators (URLs) are placed within HTML documents **301** accessible to the Web server **110**. These URLs direct browsers to generate requests for media to the system **100**. The system processes such URLs by interpreting the proprietary tags, executing the indicated image generation procedures on the original media **200**, and returning derivative Web-safe media to client browsers **120***a*–**120***d* via the Internet **130**. Additionally, such generated media is cached on the image server **100** and, therefore need not be regenerated for subsequent requests.

Web site administration according to another preferred embodiment of the invention is described with reference to FIG. **19**. FIG. **19** is a schematic diagram showing the components of Web site administration according to a preferred embodiment of the invention, whereby Web site administration is simplified. The preferred embodiment provides, but is not limited to the following services: asset management, automatic image manipulation, automatic image conversion, automatic image upload, automatic image customization based on browser characteristics, automatic disk management, automatic control of proxy caching, and image delivery **501**.

FIG. **20** is a simple overview showing components of the system according to a preferred embodiment of the invention. HTML pages with proprietary URL tags **301** describe how referenced media therein is to be manipulated for Web. Browsers **120** send such tags to the image system **100** as media requests. A server **2000** within the image system **100** receives the media requests, decodes the URL tags, and retrieves any media that already exists in the media caching system **2010**. Non-existent media is subsequently generated by a media creation system **2020** using original media **2050** stored in a media repository **2040** and using content generation procedures **2030**.

The Image System.

Following is a detailed description of the preferred embodiment of the invention with reference to FIG. **21** below.

The system receives a request for media through a URL containing proprietary tags for controlling image generation. The system parses this URL to determine the content generation procedure to execute, input to the content generation procedure, post-processing directives for, for example, zoom/pan/slice, browser properties, and any cache control directives. Such data is handed to a media caching subsystem that returns the requested image if found. If the image is not found, the information is handed to the media generation subsystem that executes the specified content generation procedure to produce a derivative image. The media generation subsystem returns one or more images to the media caching system for subsequent reuse.

The media caching subsystem is a mechanism for associating final or intermediate derivative media with the procedure, input, and user characteristics used to generate said media, specified through proprietary tags within the requested URL. This system may be implemented using a database, file system, or any other mechanism having capability to track such associations.

The media generation subsystem executes a primary content generation procedure to produce a derivative image whose identifier is provided to the media caching subsystem. This derivative image is composed of one or more original images acquired from the media repository. This media is then passed to the dynamic image content system, if necessary, to generate a subsequent derivative media suitably modified for the needs of zooming, panning, or slice. The resulting media is passed to the user profile system where it is again modified to account for any specific user browser characteristics specified using the proprietary URL tags. This media is then returned to the browser, along with any cache control directives encoded within the URL, and its identifier is passed to the media caching system for subsequent retrieval.

The dynamic content system operates on intermediate derivative images to generate image subsets or scalings used by Web site designers to implement zooming in on an image, panning across an image, slicing an image into parts, and the like for special Web page effects. The input to this system is cached by the media caching system such that the intermediate image need not be regenerated.

The user profile system operates on the final image about to be returned to the browser and may modify the image to account for individual needs of Web site users. The designer of a site is able to implement freely custom post-processing of images to meet the specific needs of their clients.

FIG. **21** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention. Original media **200** is created and placed into the system **100** in a media repository **2040**. A content generation procedure **2140** is created with instructions on how the media is to be transformed to create the desired Web page content. An HTML page **301** is created for the Web site comprising the system **100**, the page containing one or more URLs directing a browser **120** to request the specified content generation procedure **2140** from the system **100** using input parameters specified with proprietary tags encoded within the URL. The browser **120** requests the Web page **301** from the Web site **110**. Upon receipt of the page **301**, the browser contacts the system **100** requesting media specified in the URL. The system parses the URL **2100** to determine the content generation procedure **2140** to execute, any corresponding input parameters to be used by such procedure, any dynamic content processing **2150** to be performed by dynamic media procedures, any user profile information **2160** to be used to modify the resulting image, and any cache control HTTP headers **2190** the site instructs to accompany the resulting image.

The parser generates a unique primary lookup key **2110** for the specified resulting media. If the key corresponds to an existing generated media **2180**, such media is returned immediately to the browser **120** through a media cache **2120**, and the transaction is complete. Otherwise, a media generation occurs. In the case of media generation requiring dynamic content processing, a unique secondary lookup key corresponding to intermediate media is generated **2130**. If intermediate media **2170** corresponding to this key is found, such media is passed directly to the dynamic media content system **2150** having dynamic media procedures, wherein appropriate action is taken to generate the required derivative from the intermediate media data. A unique key is generated for the derivative **2130** and passed to the media caching system **2120**. If the media caching system finds no such intermediate image, such intermediate image is generated according to instructions specified by the content generation procedure, cached by the media cache system **2120** as a secondary cached media **2170**, and passed to the dynamic media system **2150**. Again, appropriate action is taken to generate the required derivative from the intermediate image data.

The resulting image after any dynamic media processing is complete, is checked to ensure that the image is in a valid Web image format. If not, the image is automatically converted into a valid format.

The final media is passed to a user profile system **2160** wherein browser characteristics specified through proprietary tags within the URL are inspected, and appropriate modification to the media is performed, based on such characteristics. The resulting image is handed to the media cache system **2120** for caching and returned to the browser **120**.

FIG. **22** shows a flowchart of the content generation procedure according to a preferred embodiment of the invention. A URL containing proprietary tags (**2200**) is parsed (**2210**) to determine the content generation procedure to execute, any dynamic modifications to the media, user profile characteristics, and proxy-cache control. A unique final lookup key is generated for the media (**2220**) and the media cache is checked (**2230**). If the indicated media exists, control passes to proxy-cache control (**2290**) and the media is delivered to the browser (**2295**). Otherwise, dynamic media system tags are separated from content generation control tags (**2240**) and a unique intermediate image lookup key is generated (**2250**). The cache is then checked for such intermediate media (**2261**). If such intermediate media is found, it is used directly for dynamic processing, if required. Otherwise, content is generated (**2262**) and cached (**2263**), and the result is evaluated for dynamic processing (**2270**). If dynamic processing is required, the media is operated upon by the dynamic content generator (**2271**), otherwise it is evaluated for valid content type (**2272**). If the content type is invalid, the media is automatically converted to a valid type (**2273**). The resulting image is then customized by the user profiling system (**2280**) for specified browser or client attributes. Finally, any cache-control directives specified are attached to the response (**2290**) and the media is delivered to the browser (**2295**).

FIG. **23** is a flow chart showing an authoring process according to a preferred embodiment of the invention. The process starts (**2300**) when a user adds an original graphic or other media (**2310**) to the system. The author then creates a content generation procedure (**2320**) containing instruction on how the original media should be processed to generate the desired Web page content. The user then creates an HTML document (**2330**) that refers to that image by using a URL pointing to a content generation procedure on the image server. The system ends (**2340**). The authoring subsystem assists the Web site designer with choosing parameters and with designing the content generation procedure such that the desired Web site graphic is obtained.

It should be appreciated that differences exist between specifying an image with a URL and requesting an image using a content creation process that interprets proprietary parameters encoded within a URL. That is, URLs allow Web site designers to load specific graphic images into a Web page. In contrast and according to the invention, URLs containing proprietary content creation tags initiate a process whereby graphic images for a site are automatically produced.

Table D below is a list of example proprietary URL tags used for content generation within the system according to the preferred embodiment of the invention. Additional tags may be added to the system as necessary.

TABLE D

Tags

f=function
    Names the content creation procedure used to generate all or part of the desired graphic.
args=arguments
    Supplies page dependent parameters used to control the content creation procedure from within the Web page.
cr=crop rectangle
    Specifies that portion of the image generated by the content generation procedure to be returned to the browser.
st=slice table
    Specifies a rectangular grid to be placed over the image produced by the content generation procedure, each portion of which can be returned to the browser.
sp=slice position
    Specifies that portion of the slice table grid placed over the image generated by the content creation procedure to be returned to the browser.
is=image size parameter
    Specifies scale factors to be applied to any portion of an image generated by any combination of a content generation procedure, arguments, crop rectangles, slice tables, and slice positions.
p=user profile string
    Specifies a user profile identifier used to modify the final image prior to returning the image to the browser, thus allowing clients to modify the image returned to the browser to account for individual browsing conditions.
c=cache control
    Specifies a proxy-cache control string to accompany the returned image within an HTTP header.

Table E below is a list of example supported content creation commands according to a preferred embodiment of the invention. Additional commands may be added as necessary.

TABLE E

Content Creation Commands

Adjust HSB
    Allows the HSB of an image to be altered.
Adjust RGB
    Allows the contrast, brightness, and color balance of an image to be altered.
Colorize
    Alters the hue of the pixels in the image to that of the specified color.
Brush Composite
    Composites the specified brush image onto the current target image.
Convert
    Converts the rasters to the specified type/bit-depth.
Crop
    Crops the media to the specified size.
Dropshadow
    Adds a drop shadow to the image, based on the alpha-channel.
Equalize
    Performs an equalization on the relevant components of the media.
FixAlpha
    Adjusts the RGB components of an image relative to its alpha-channel.
Flip
    Flips the media vertically or horizontally.
Glow
    Produces a glow or halo around the image
Load
    Loads in a media from the specified file.
Normalize
    Similar to equalize, but for audio.
Reduce
    Reduces the image to a specified palette.

TABLE E-continued

Content Creation Commands
Rotate Rotates the media clockwise by the specified angle in degrees.
Save Saves the media to the specified file.
Scale Scales the media to the specified size.
SetColor Allows the background color, foreground color, and transparency state of the media to be set.
Text Drawing Composites the specified text onto the image.
Text Making This command, instead of compositing text onto the target, creates a new image that just encloses the text.
Zoom Zooms in on a specified portion of the media, and fits it to the specified size. Effectively this constitutes a crop followed by a scale.

Table F below lists comprises some, and is not limited to all major features of a preferred embodiment of the invention. Additional features may be added as necessary.

TABLE F

System Features
Reads and writes various file formats;
Supports many image processing operations;
Dynamically creates Web images from original assets;
Dynamically creates thumbnail images;
Dynamically and efficiently creates images that can be panned, zoomed, or sliced from original assets without Browser plugins;
Automatically propagates changes in original assets throughout the Web site;
Uses an intelligent caching mechanism for both final and intermediate graphics, comprising:
Clean up cache on demand;
Eliminates orphaned Web files; and
Optimizes Web server cache by providing most recent images;
Renders True-Type fonts on server instead of browser;
Uses intelligent scaling of line drawings;
Allows Web designers to manipulate images using a combination of content generation procedures and proprietary URL tags;
Preserves original image assets;
Optimizes Web server traffic by adjusting the bandwidth of graphics;
Optimizes images for client connection speed;
Allows clients to specify the quality of images on a Web site:
Allow site-specific customized image optimizations for a variety of purposes; and
Allows Web designers to dynamically create images by manipulating proprietary URL tags in applications.

Accordingly, although the invention has been described in detail with reference to a particular preferred embodiment, persons possessing ordinary skill in the art to which this invention pertains will appreciate that various modifications and enhancements may be made without departing from the spirit and scope of the claims that follow.

What is claimed is:

1. A computer implemented method for generating and delivering Web page content comprising media, said method comprising:

parsing a URL containing proprietary tags to determine:
a content generation procedure to execute and corresponding input to said procedure;
dynamic modifications to perform on said media;
user profile characteristics; and
proxy-cache control;

generating a unique final lookup key for said media;
checking a media cache, using said final lookup key;
if said media exists in said media cache, then
passing control to said proxy-cache control; and
delivering said media;
if said media does not exist in said media cache, then
separating dynamic media system tags from content generation control tags; and
generating a unique intermediate image lookup key;
checking said media cache using said intermediate image lookup key;
if said intermediate media exists in said media cache, then using said intermediate media for further processing;
if said intermediate media does not exist in said media cache, then
generating and caching said media, using said intermediate image lookup key;
determining if dynamic processing is required and, if affirmative, then
operating upon said media by a dynamic content generator,
determining if content type is valid and, if negative, then
converting said media automatically to a valid type;
customizing said media for specified browser or client attributes using a user profiling system;

attaching any specified cache-control directives to a response; and

delivering said media.

2. The method of claim 1, wherein dynamic processing comprises directives for zoom, pan, and slice.

3. An apparatus for generating Web-safe media, using an HTML page having a URL, said URL having encoded proprietary tags, said apparatus comprising:

a server for receiving media requests and delivering said Web-safe media;

a media repository for storing original media;

a content generation procedure containing instructions for transforming said original media into said Web-safe media;

a URL tag parser for determining:

said content generation procedure to execute and any corresponding input parameters to be used by said procedure for generating a primary media to be cached;

dynamic content processing to be performed, if necessary by dynamic media procedures;

user profile information, if any to be used for modification of a resulting image; and

cache control HTTP headers, if any to accompany said resulting image;

a unique primary lookup key generated by said parser and associated with said primary cached media, wherein said primary cached media, if existing, is delivered as said Web-safe media through a media cache; and

a unique secondary lookup key corresponding to intermediate media requiring said dynamic media processing by said dynamic media procedures, thereby generating corresponding derivative intermediate media, and said unique secondary lockup key corresponding to said derivative intermediate media stored in a secondary media cache.

4. The apparatus of claim 3, wherein any of said original media, said content generation procedure, and said proprietary URL tags are modified, thereby rendering an automatically updated Web-safe media.

5. The apparatus of claim 5, wherein said apparatus is configured in parallel to a Web server infrastructure, said existing Web server infrastructure providing said media requests.

6. The apparatus of claim 5, wherein said apparatus is configurable to work either on-site or off-site to said Web server infrastructure.

7. The apparatus of claim 3, wherein said primary cached media is composed of one or more original media images acquired from said media repository.

8. The apparatus of claim 3, wherein said derivative intermediate media is suitably modified for the needs of zooming, panning, and slicing.

* * * * *

Exhibit C

US008381110B2

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

(10) **Patent No.: US 8,381,110 B2**
(45) **Date of Patent: Feb. 19, 2013**

(54) **AUTOMATED MEDIA DELIVERY SYSTEM**

(75) Inventors: **Sean Barger**, Mill Valley, CA (US); **Steve Johnson**, Mill Valley, CA (US); **Matt Butler**, Beaverton, OR (US); **Jerry Destremps**, Sausalito, CA (US); **David Pochron**, Cambridge, WI (US); **Trent Brown**, San Anselmo, CA (US)

(73) Assignee: **Equilibrium**, Sausalito, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 534 days.

(21) Appl. No.: **12/238,842**

(22) Filed: **Sep. 26, 2008**

(65) **Prior Publication Data**

US 2009/0089422 A1 Apr. 2, 2009

Related U.S. Application Data

(60) Division of application No. 12/173,747, filed on Jul. 15, 2008, which is a division of application No. 11/269,916, filed on Nov. 7, 2005, now abandoned, which is a continuation-in-part of application No. 09/929,904, filed on Aug. 14, 2001, now Pat. No. 6,964,009, which is a continuation of application No. 09/425,326, filed on Oct. 21, 1999, now Pat. No. 6,792,575.

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

(52) **U.S. Cl.** **715/736**; 715/201; 715/224; 715/238; 715/249; 715/273; 715/704; 715/733; 715/738; 715/748

(58) **Field of Classification Search** 709/219, 709/236, 224; 715/733, 738, 201, 224, 238, 715/249, 273, 704, 736, 748
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,088,052	A	2/1992	Spielman et al.
5,355,472	A	10/1994	Lewis
5,442,771	A	8/1995	Filepp et al.
5,530,852	A	6/1996	Meske, Jr. et al.
5,701,451	A	12/1997	Rogers et al.
5,708,845	A	1/1998	Wistendahl et al.
5,710,918	A	1/1998	Lagarde et al.
5,737,619	A	4/1998	Judson
5,745,908	A	4/1998	Anderson et al.
5,758,110	A	5/1998	Boss et al.
5,761,655	A	6/1998	Hoffman
5,793,964	A	8/1998	Rogers et al.
5,819,261	A	10/1998	Takahashi et al.
5,822,436	A	10/1998	Rhoads
5,845,084	A	12/1998	Cordell et al.
5,845,279	A	12/1998	Garofalakis et al.
5,845,299	A	12/1998	Arora et al.
5,860,068	A	1/1999	Cook

(Continued)

FOREIGN PATENT DOCUMENTS

AU	A-53031/98	8/1996
EP	0747842	12/1996

(Continued)

OTHER PUBLICATIONS

Sakaguchi, et al.; "A browsing tool for multi-lingual documents for users without multi-lingual fonts"; 1996; ACM International Conference on Digital Libraries, pp. 63-71.

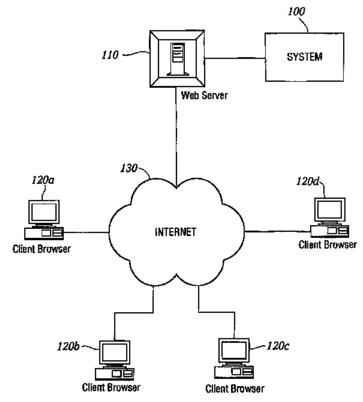
(Continued)

Primary Examiner — Cesar Paula
Assistant Examiner — David Faber
(74) *Attorney, Agent, or Firm* — Michael A. Glenn; Glenn Patent Group

(57) **ABSTRACT**

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed by an author within URLs embedded within Web documents.

10 Claims, 23 Drawing Sheets



U.S. PATENT DOCUMENTS

5,860,073	A	1/1999	Ferrel et al.	
5,861,881	A	1/1999	Freeman et al.	
5,862,325	A	1/1999	Reed et al.	
5,864,337	A	1/1999	Marvin	
5,870,552	A	2/1999	Dozier et al.	
5,880,740	A	3/1999	Halliday et al.	
5,890,170	A	3/1999	Sidana	
5,895,476	A	4/1999	Orr et al.	
5,895,477	A	4/1999	Orr et al.	
5,903,892	A	5/1999	Hoffert et al.	
5,937,160	A	8/1999	Davis et al.	
5,943,680	A	8/1999	Ohga et al.	
5,956,737	A	9/1999	King et al.	
6,009,436	A	12/1999	Motoyama et al.	
6,311,185	B1	10/2001	Markowitz et al.	
6,456,305	B1	9/2002	Qureshi et al.	
6,463,445	B1	10/2002	Suzuki et al.	
6,483,851	B1	11/2002	Neogi	
6,484,149	B1	11/2002	Jemmes et al.	
6,563,517	B1	5/2003	Bhagwat et al.	
6,591,280	B2	7/2003	Orr	
6,623,529	B1	9/2003	Lakritz	
6,909,708	B1	6/2005	Krishnaswamy et al.	
6,938,073	B1 *	8/2005	Mendhekar et al.	709/217
7,284,201	B2 *	10/2007	Cohen-Solal	715/738
7,313,361	B2	12/2007	Steelberg et al.	
7,406,434	B1	7/2008	Chang et al.	
7,477,688	B1 *	1/2009	Zhang et al.	375/240
7,673,063	B2 *	3/2010	Xie et al.	709/231
2003/0225568	A1	12/2003	Salmonsen	
2004/0025176	A1	2/2004	Franklin et al.	
2005/0255852	A1	11/2005	Steelberg et al.	
2005/0278794	A1 *	12/2005	Leinonen et al.	726/32
2006/0015580	A1 *	1/2006	Gabriel et al.	709/219
2006/0127059	A1 *	6/2006	Fanning	386/125
2007/0061198	A1	3/2007	Ramer et al.	
2008/0155230	A1	6/2008	Robbins et al.	
2008/0195938	A1	8/2008	Tischer et al.	
2008/0205389	A1	8/2008	Fang	
2008/0207182	A1	8/2008	Maharajh et al.	
2009/0013347	A1	1/2009	Ahanger et al.	
2009/0070485	A1	3/2009	Barger et al.	
2009/0240569	A1	9/2009	Ramer et al.	
2009/0254672	A1	10/2009	Zhang	
2010/0153495	A1	6/2010	Barger et al.	

FOREIGN PATENT DOCUMENTS

EP	0782085	7/1997
EP	0818907	1/1998
EP	0843276	5/1998
EP	0876034	11/1998
EP	0883068	12/1998
EP	0886409	12/1998
EP	0895171	2/1999
EP	0926607	6/1999
EP	0949571	10/1999
WO	WO 97/49252	12/1997
WO	WO 98/40842	9/1998
WO	WO 98/43177	10/1998

OTHER PUBLICATIONS

Zaiane, et al.; “Mining multimedia data”; Nov. 1998; ACM Conference of the Center for Advanced Studies on Collaborative research, pp. 1-18.

Bulterman, Dick.C.A.; *Models, Media and Motion: Using the Web to Support Multimedia Documents*; Proceedings of 1997 Intnl Conf on Multimedia Modeling; p. 17-20; Nov. 1997; Singapore.

Mohler, J.L.; *Migrating Course Materials to the World Wide Web: A Case Study of the Department of Technical Graphics at Purdue University*; Computer Networks and ISDN Systems; vol. 30, Issues 20-21, p. 1981-1990; Nov. 12, 1988.

Dobson, R.; *Animating Your Web Pages with Direct Animation*; Web Techniques; vol. 3, No. 6, p. 49-52; Jun. 1998.

Berinstein, Paula; “The Big Picture; Text and Graphics on UMI’s ProQuest Direct: The Best (Yet) of Both Words”; Mar. 1997; retrieved on Mar. 23, 2004 from website: http://www.infotoday.com/online/MarOL97/picture3.html.

McNeil, Sara; Research Interests; retrieved on Mar. 18, 2004 from website: http//www.coe.uh.edu/-smcneil/research.htm.

Tables of Contents service for Computers & Geosciences; Copyright 1997; Computers and GeoSciences, vol. 23, Issue 5, retrieved on Mar. 18, 2004 from website: http://library.iem.ac.ru/comp&geo/00983004/sz977014.html.

* cited by examiner

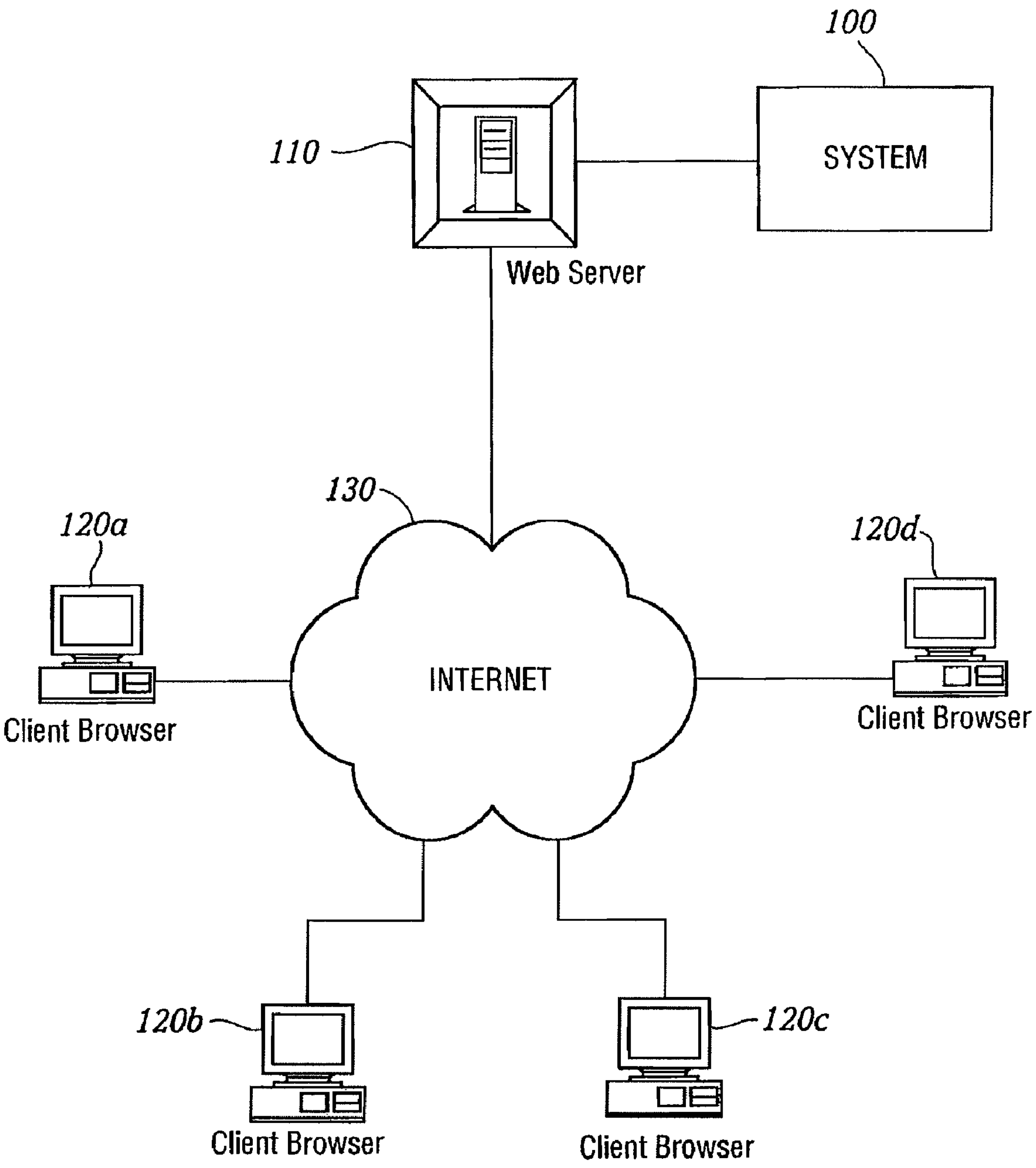

100
SYSTEM
110
Web Server
130
INTERNET
120a
Client Browser
120d
Client Browser
120b
Client Browser
120c
Client Browser

FIG. 1

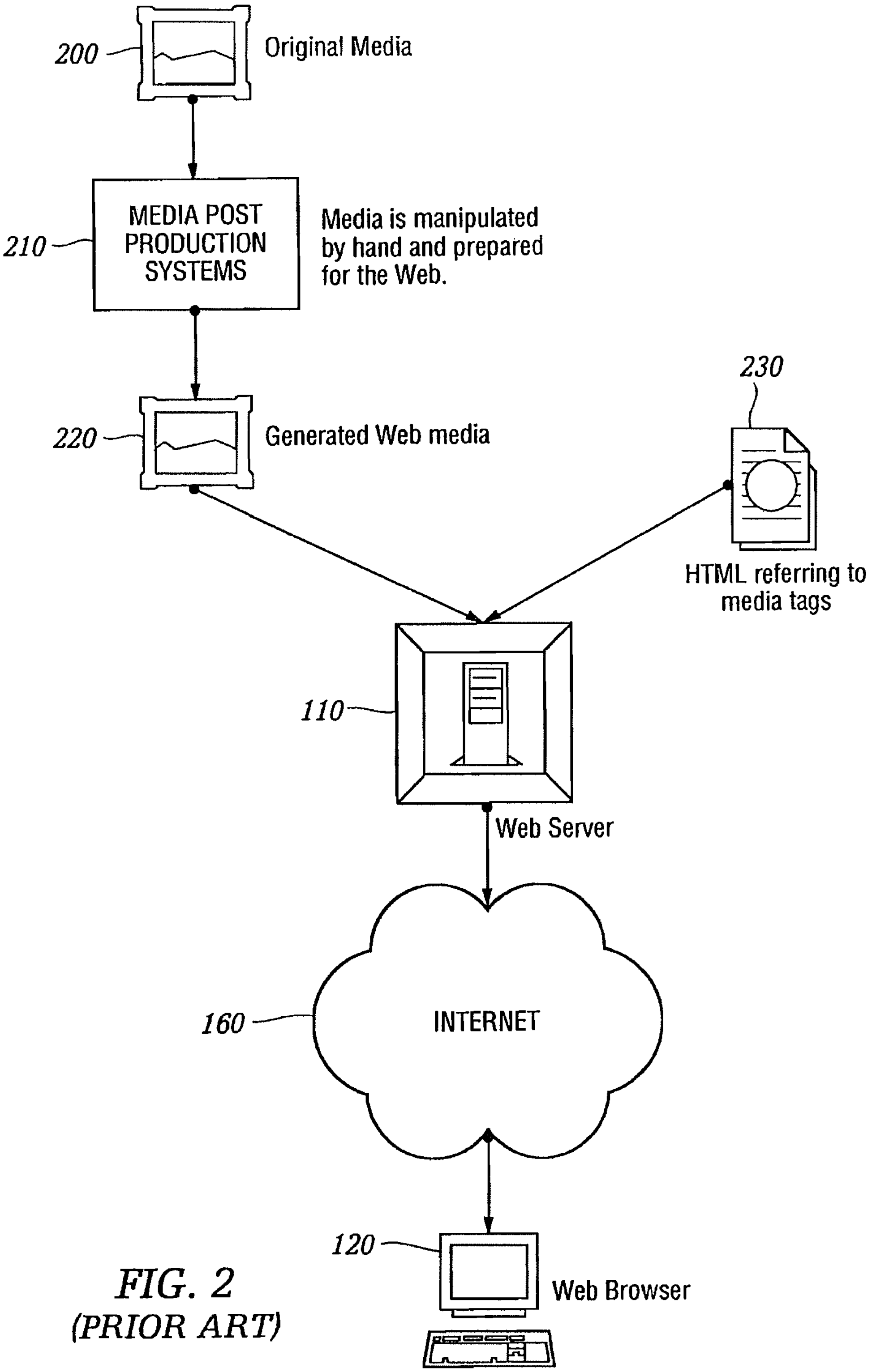

200
Original Media
210
MEDIA POST PRODUCTION SYSTEMS
Media is manipulated by hand and prepared for the Web.
230
220
Generated Web media
HTML referring to media tags
110
Web Server
160
INTERNET
120
Web Browser

FIG. 2
(PRIOR ART)

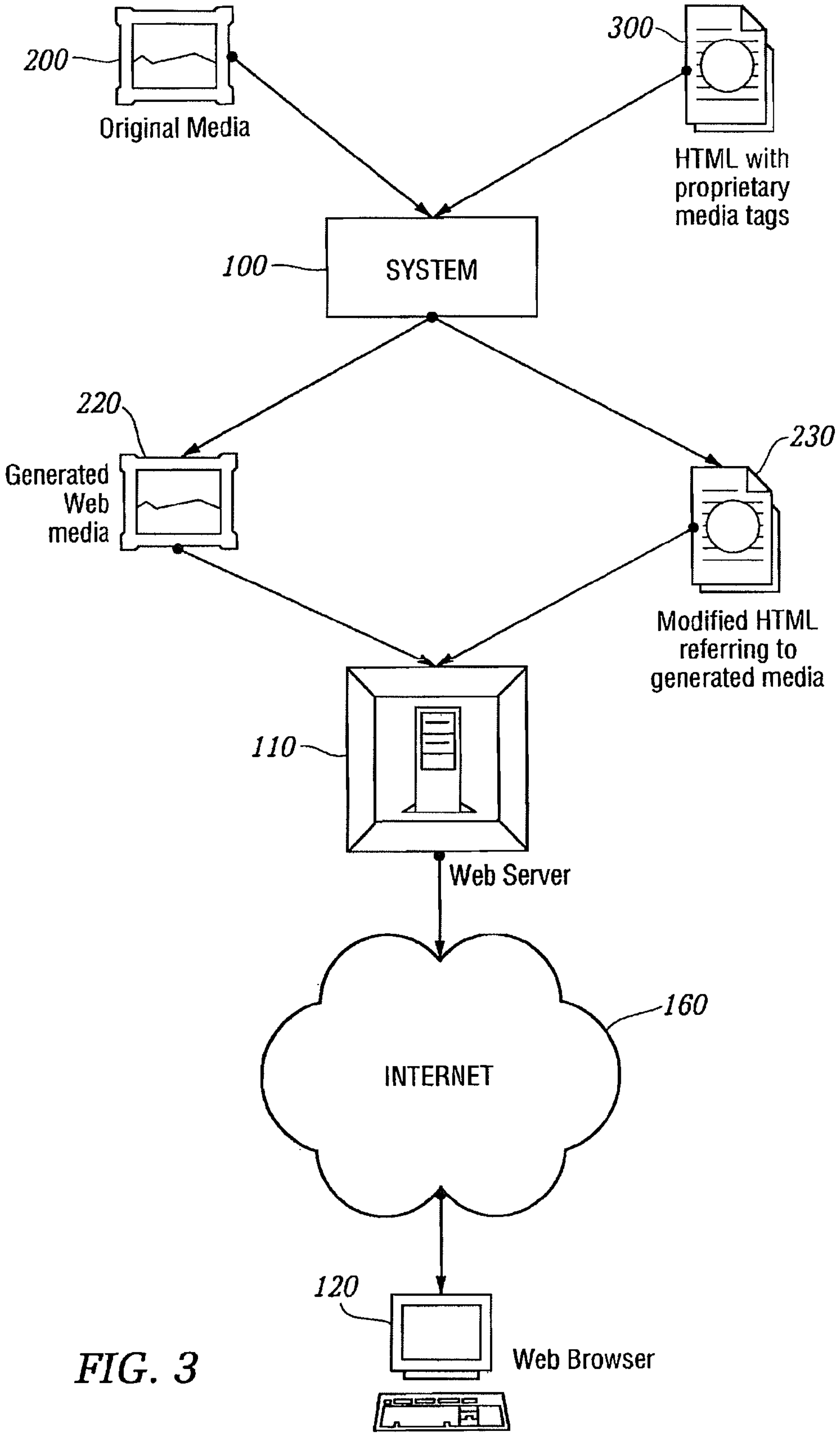

200
Original Media
300
HTML with proprietary media tags
100
SYSTEM
220
Generated Web media
230
Modified HTML referring to generated media
110
Web Server
160
INTERNET
120
Web Browser

FIG. 3

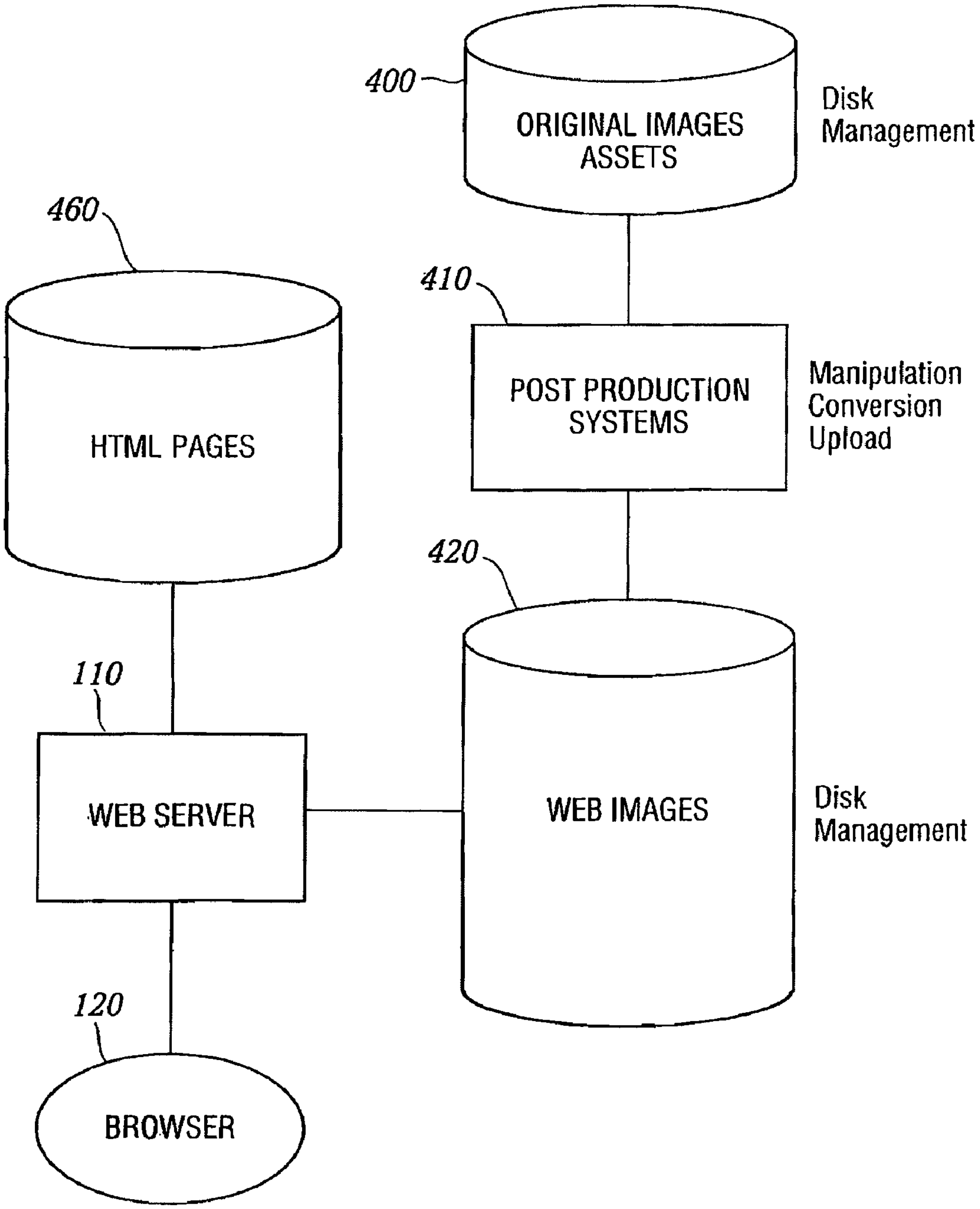


FIG. 4
(PRIOR ART)

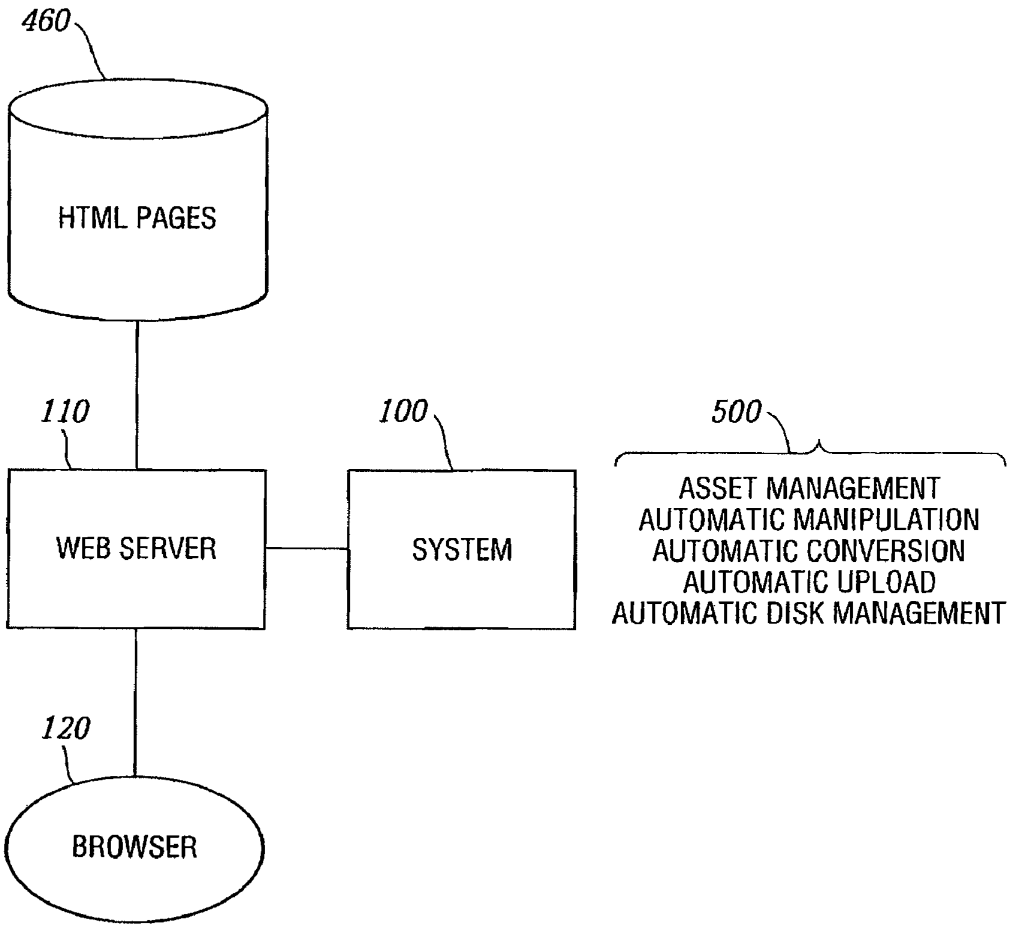

460
HTML PAGES
110
WEB SERVER
100
SYSTEM
500
ASSET MANAGEMENT
AUTOMATIC MANIPULATION
AUTOMATIC CONVERSION
AUTOMATIC UPLOAD
AUTOMATIC DISK MANAGEMENT
120
BROWSER

FIG. 5

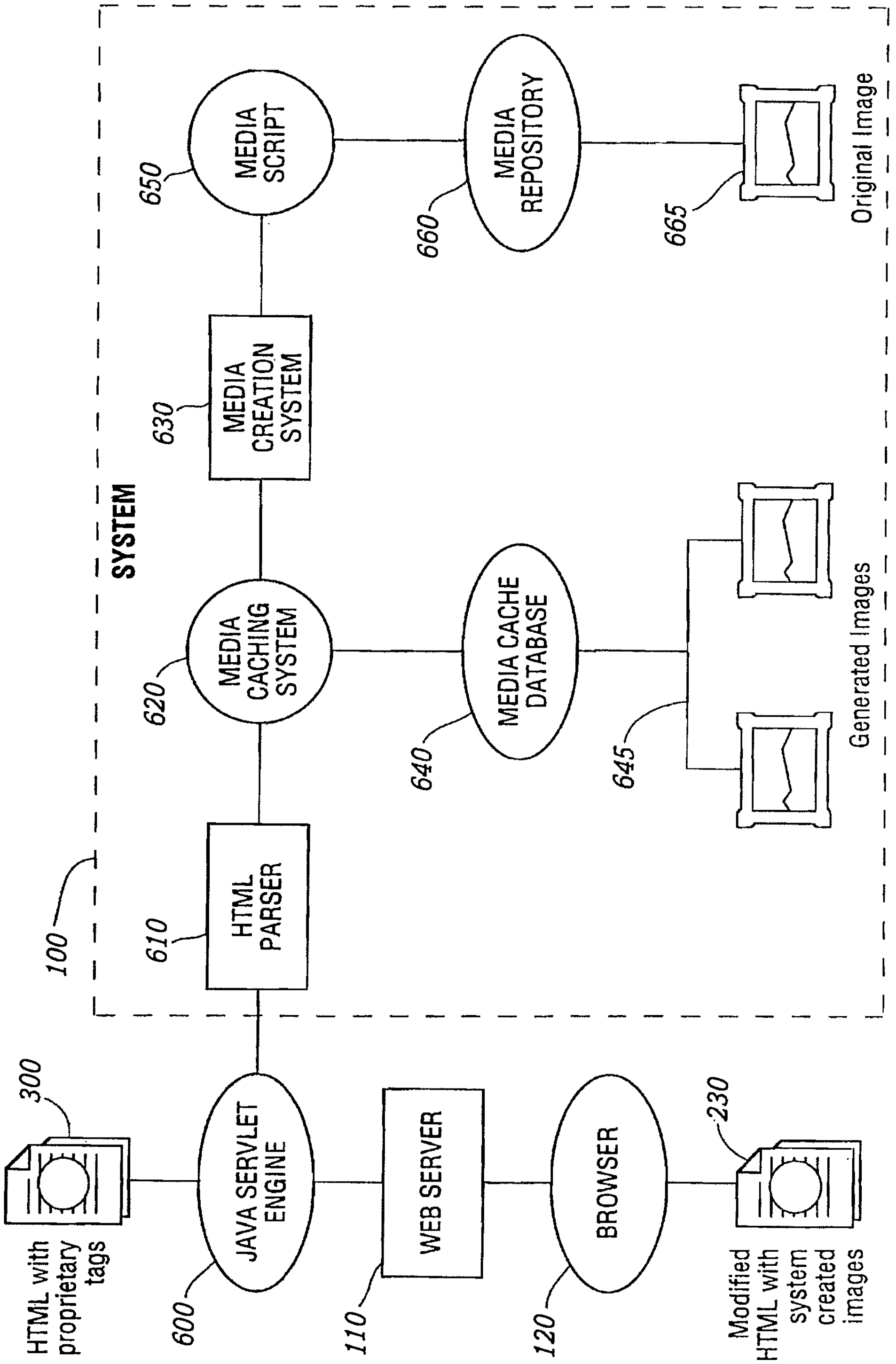

100
SYSTEM
610
HTML PARSER
620
MEDIA CACHING SYSTEM
630
MEDIA CREATION SYSTEM
650
MEDIA SCRIPT
640
MEDIA CACHE DATABASE
660
MEDIA REPOSITORY
645
Generated Images
665
Original Image
300
HTML with proprietary tags
600
JAVA SERVLET ENGINE
110
WEB SERVER
120
BROWSER
230
Modified HTML with system created images

FIG. 6

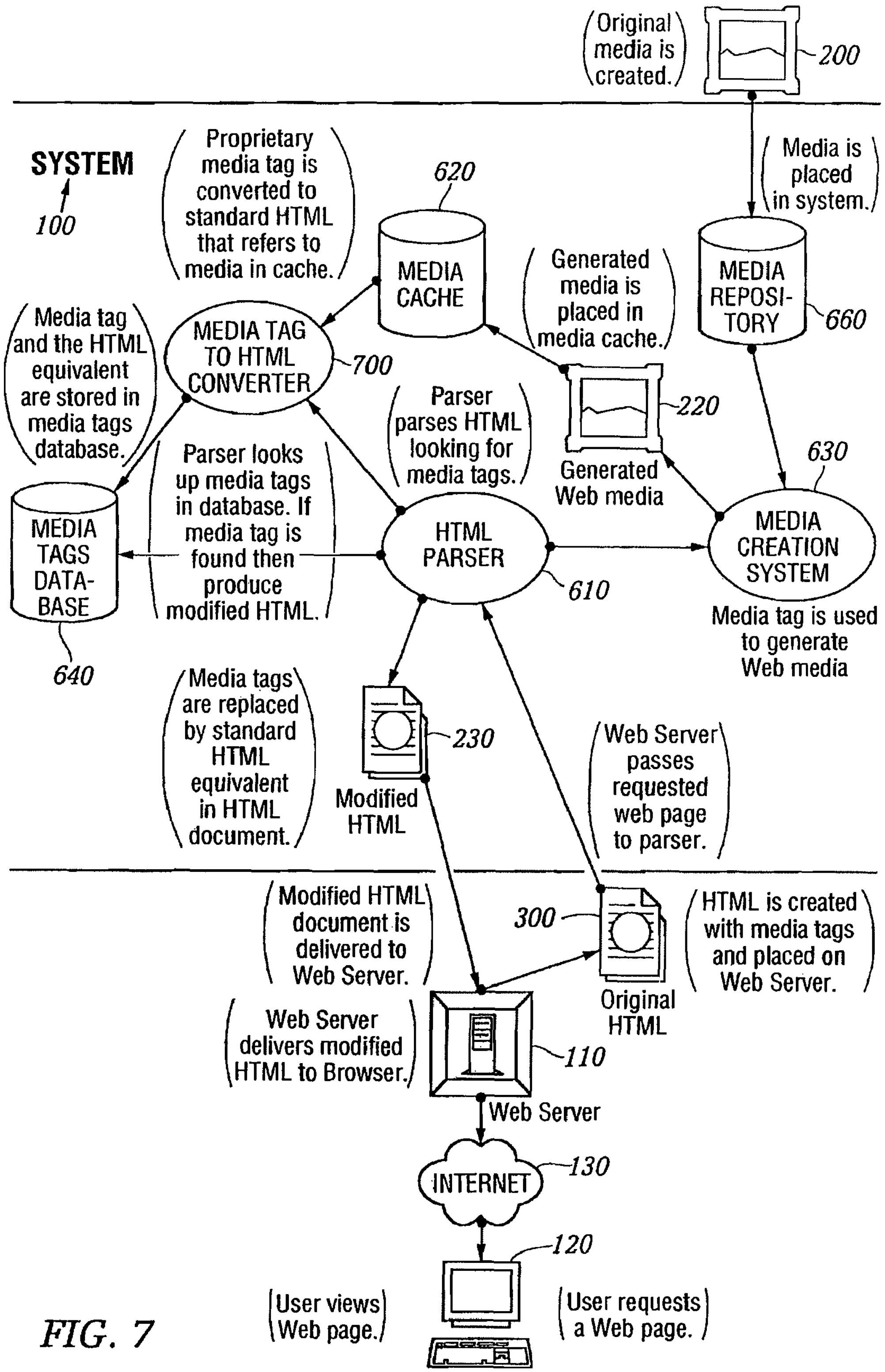

Original media is created.
200
SYSTEM
100
Proprietary media tag is converted to standard HTML that refers to media in cache.
620
MEDIA CACHE
Media is placed in system.
MEDIA REPOSITORY
660
Generated media is placed in media cache.
Media tag and the HTML equivalent are stored in media tags database.
MEDIA TAG TO HTML CONVERTER
700
Parser parses HTML looking for media tags.
220
Generated Web media
630
MEDIA TAGS DATABASE
640
Parser looks up media tags in database. If media tag is found then produce modified HTML.
HTML PARSER
610
MEDIA CREATION SYSTEM
Media tag is used to generate Web media
Media tags are replaced by standard HTML equivalent in HTML document.
230
Modified HTML
Web Server passes requested web page to parser.
Modified HTML document is delivered to Web Server.
300
Original HTML
HTML is created with media tags and placed on Web Server.
Web Server delivers modified HTML to Browser.
110
Web Server
INTERNET
130
120
User views Web page.
User requests a Web page.

FIG. 7

AUTHORING FLOWCHART

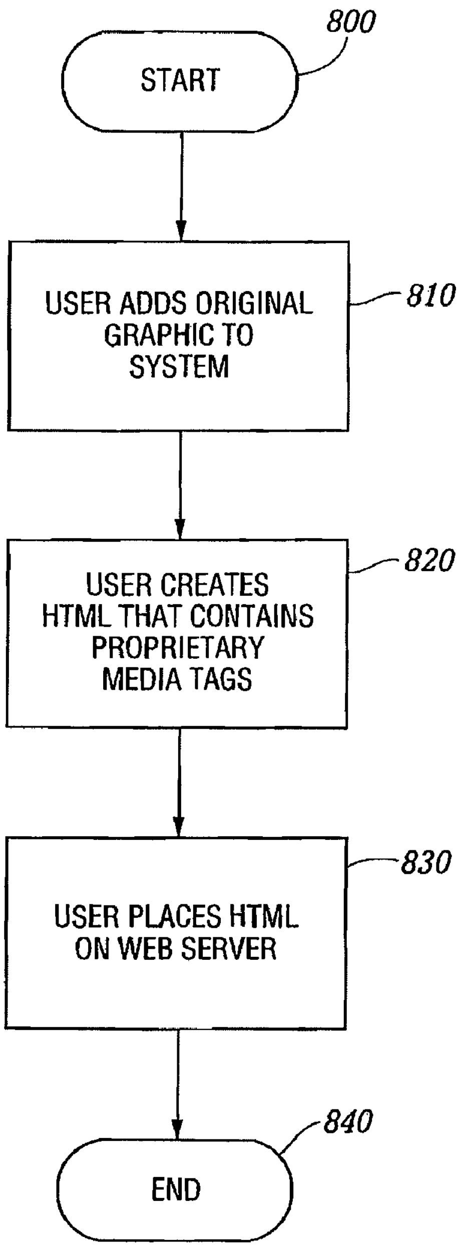

800
START
USER ADDS ORIGINAL GRAPHIC TO SYSTEM
810
USER CREATES HTML THAT CONTAINS PROPRIETARY MEDIA TAGS
820
USER PLACES HTML ON WEB SERVER
830
840
END

FIG. 8

HTML PARSING FLOWCHART

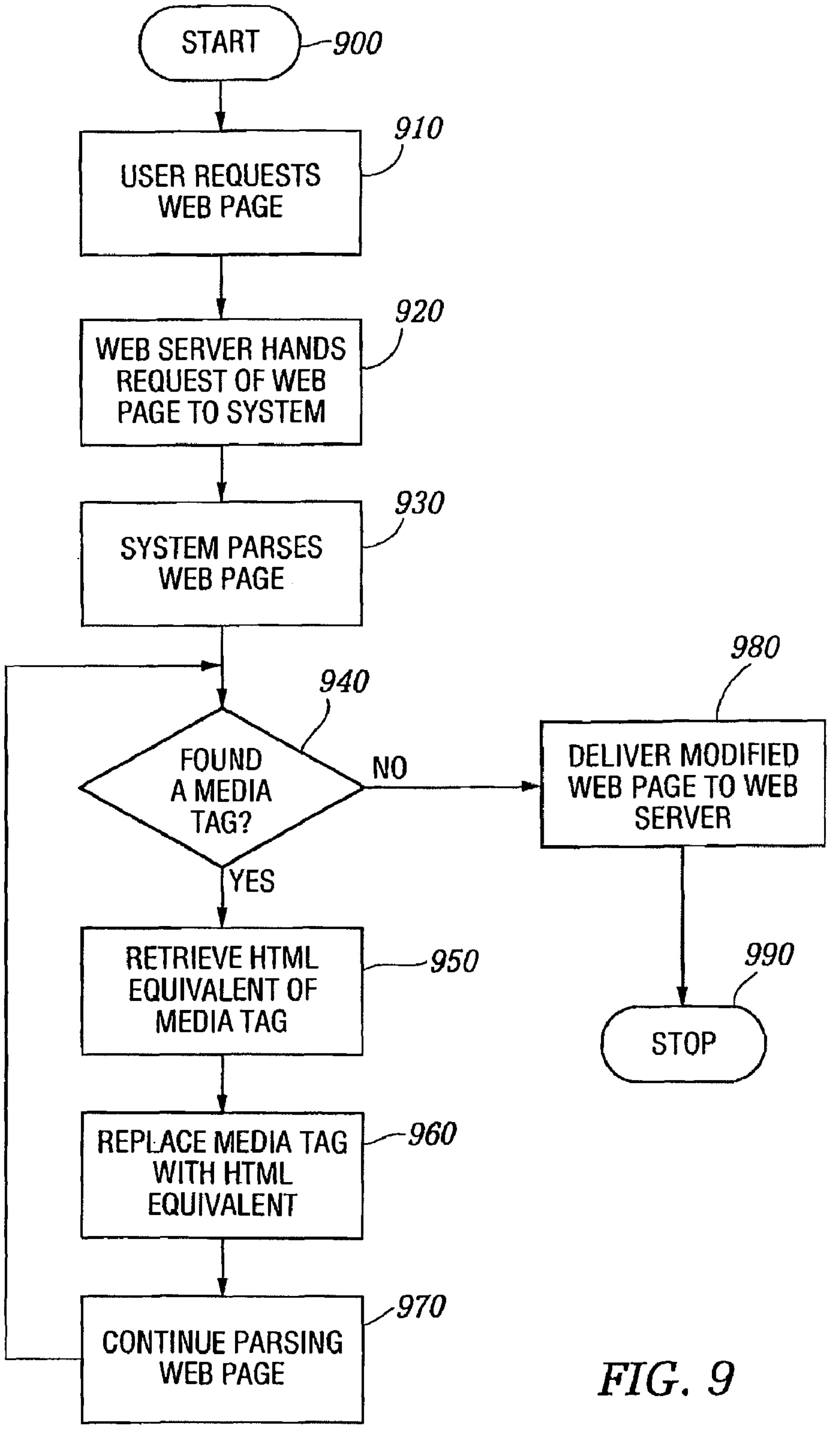

START
900
USER REQUESTS WEB PAGE
910
WEB SERVER HANDS REQUEST OF WEB PAGE TO SYSTEM
920
SYSTEM PARSES WEB PAGE
930
FOUND A MEDIA TAG?
940
NO
YES
DELIVER MODIFIED WEB PAGE TO WEB SERVER
980
STOP
990
RETRIEVE HTML EQUIVALENT OF MEDIA TAG
950
REPLACE MEDIA TAG WITH HTML EQUIVALENT
960
CONTINUE PARSING WEB PAGE
970

FIG. 9

MEDIA CREATION FLOWCHART

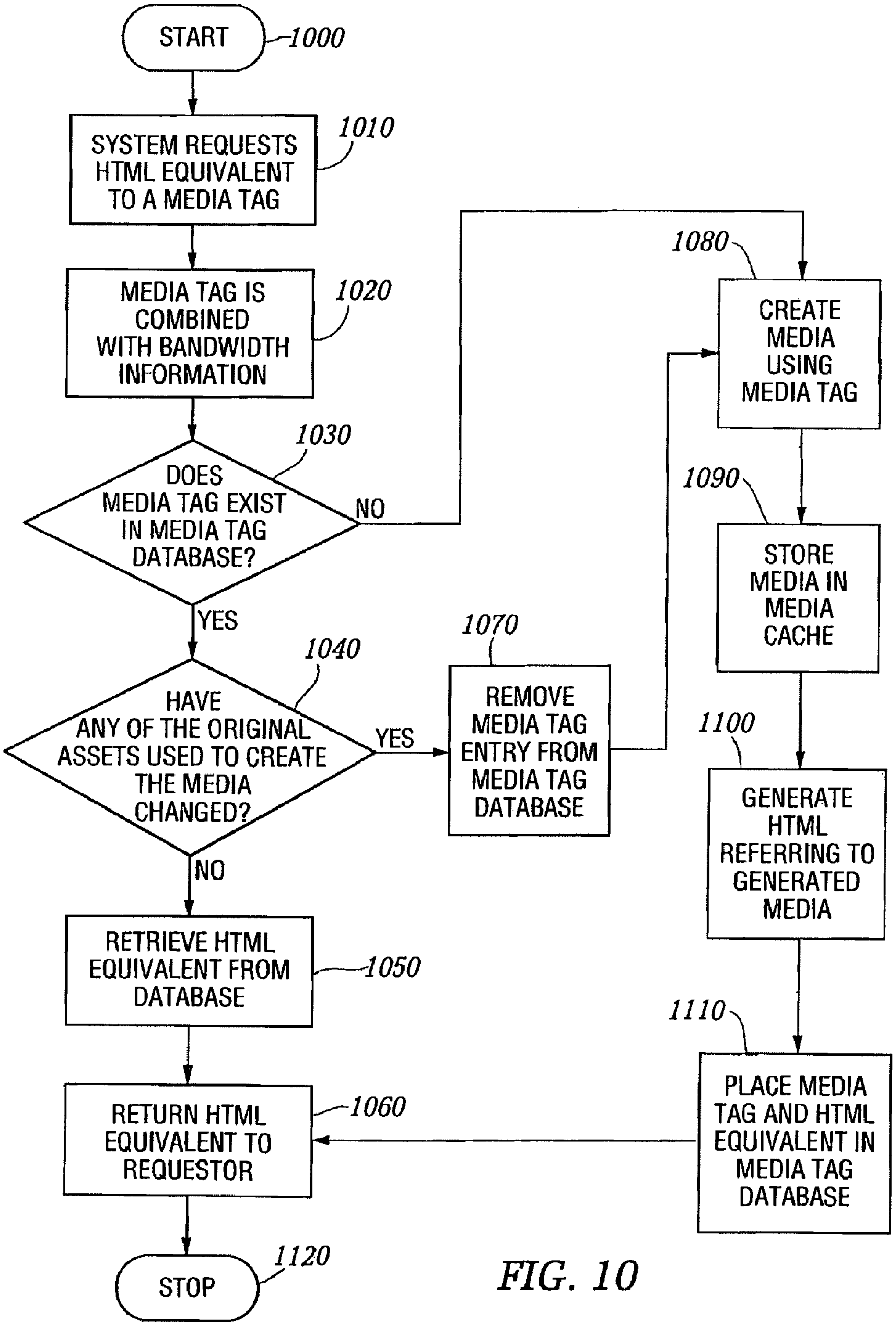
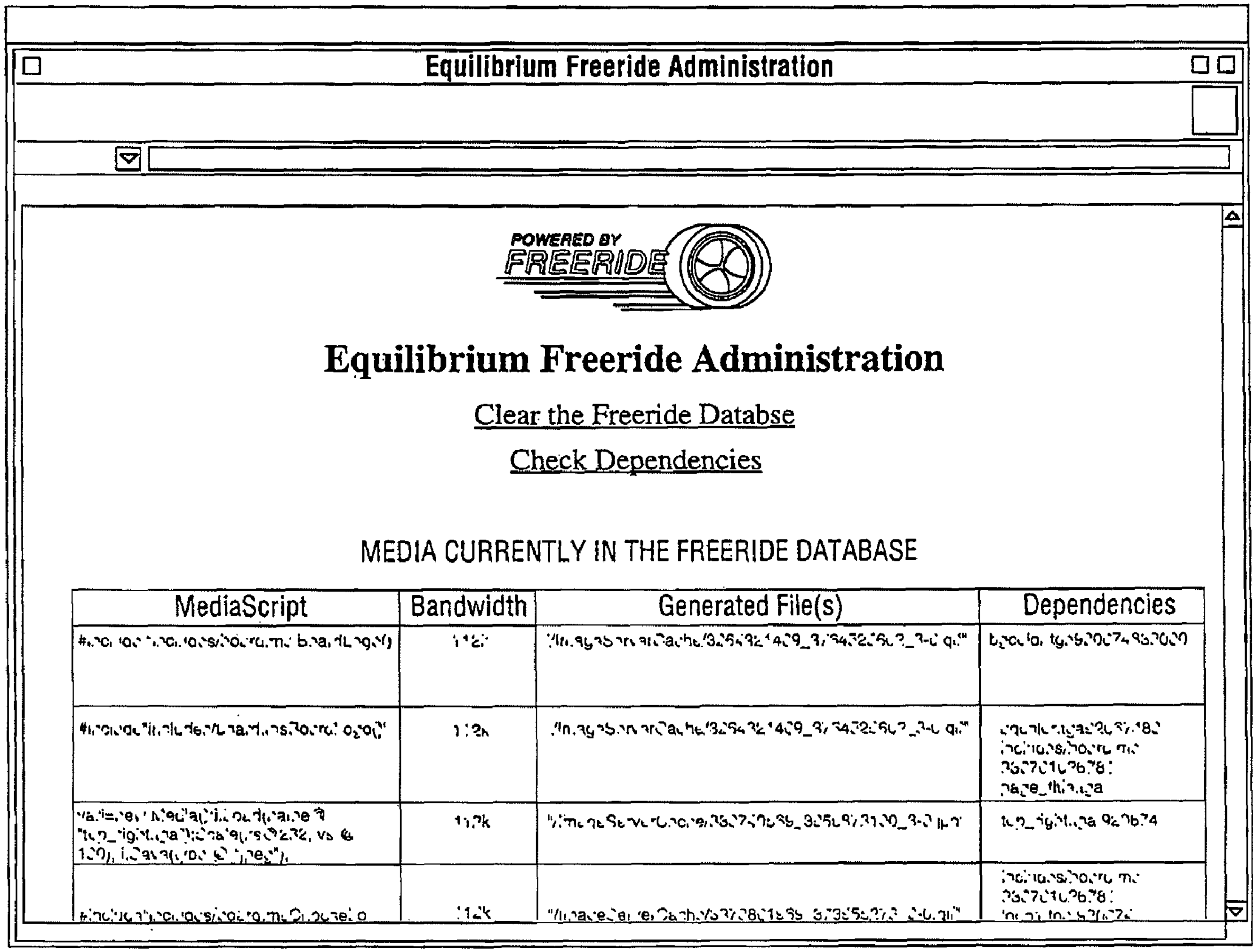

START
1000
SYSTEM REQUESTS HTML EQUIVALENT TO A MEDIA TAG
1010
MEDIA TAG IS COMBINED WITH BANDWIDTH INFORMATION
1020
DOES MEDIA TAG EXIST IN MEDIA TAG DATABASE?
1030
NO
YES
HAVE ANY OF THE ORIGINAL ASSETS USED TO CREATE THE MEDIA CHANGED?
1040
YES
NO
RETRIEVE HTML EQUIVALENT FROM DATABASE
1050
RETURN HTML EQUIVALENT TO REQUESTOR
1060
STOP
1120
REMOVE MEDIA TAG ENTRY FROM MEDIA TAG DATABASE
1070
CREATE MEDIA USING MEDIA TAG
1080
STORE MEDIA IN MEDIA CACHE
1090
GENERATE HTML REFERRING TO GENERATED MEDIA
1100
PLACE MEDIA TAG AND HTML EQUIVALENT IN MEDIA TAG DATABASE
1110

FIG. 10



Equilibrium Freeride Administration

Clear the Freeride Databse

Check Dependencies

MEDIA CURRENTLY IN THE FREERIDE DATABASE

MediaScript	Bandwidth	Generated File(s)	Dependencies
[illegible]	112k	[illegible]	[illegible]
[illegible]	112k	[illegible]	[illegible]
[illegible]	112k	[illegible]	[illegible]
[illegible]	112k	[illegible]	[illegible]

FIG. 11

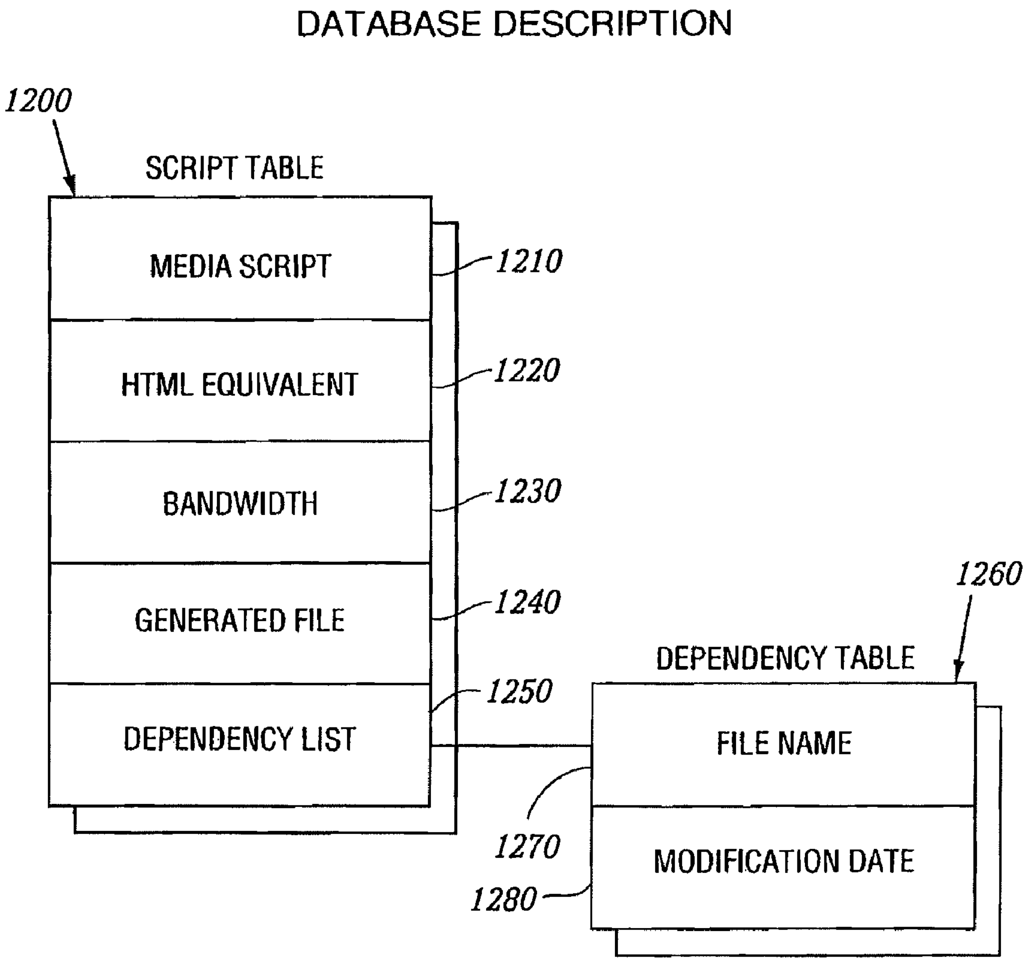

DATABASE DESCRIPTION
1200
SCRIPT TABLE
MEDIA SCRIPT
1210
HTML EQUIVALENT
1220
BANDWIDTH
1230
GENERATED FILE
1240
DEPENDENCY LIST
1250
1260
DEPENDENCY TABLE
FILE NAME
1270
MODIFICATION DATE
1280

FIG. 12

ORIGINAL IMAGES

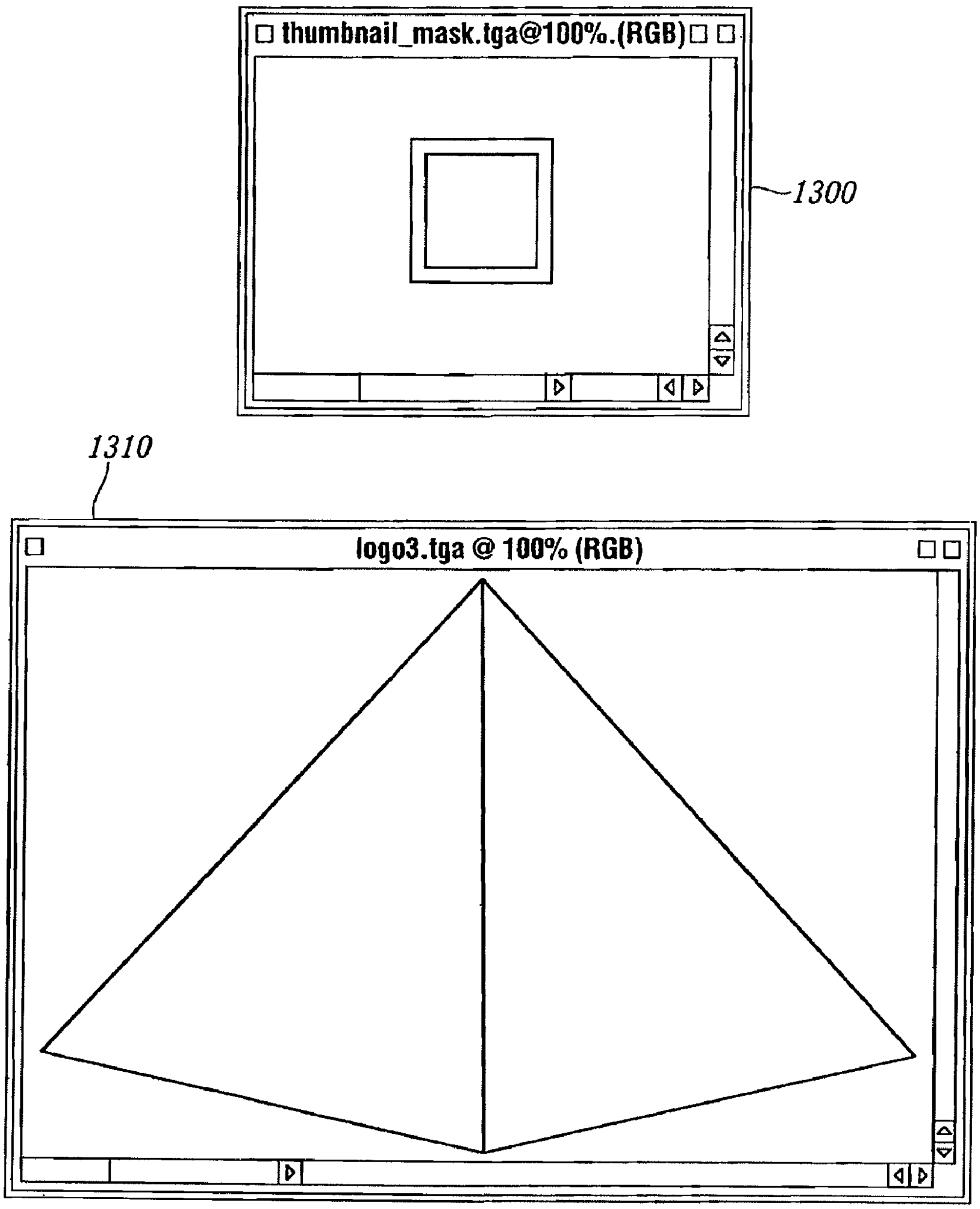
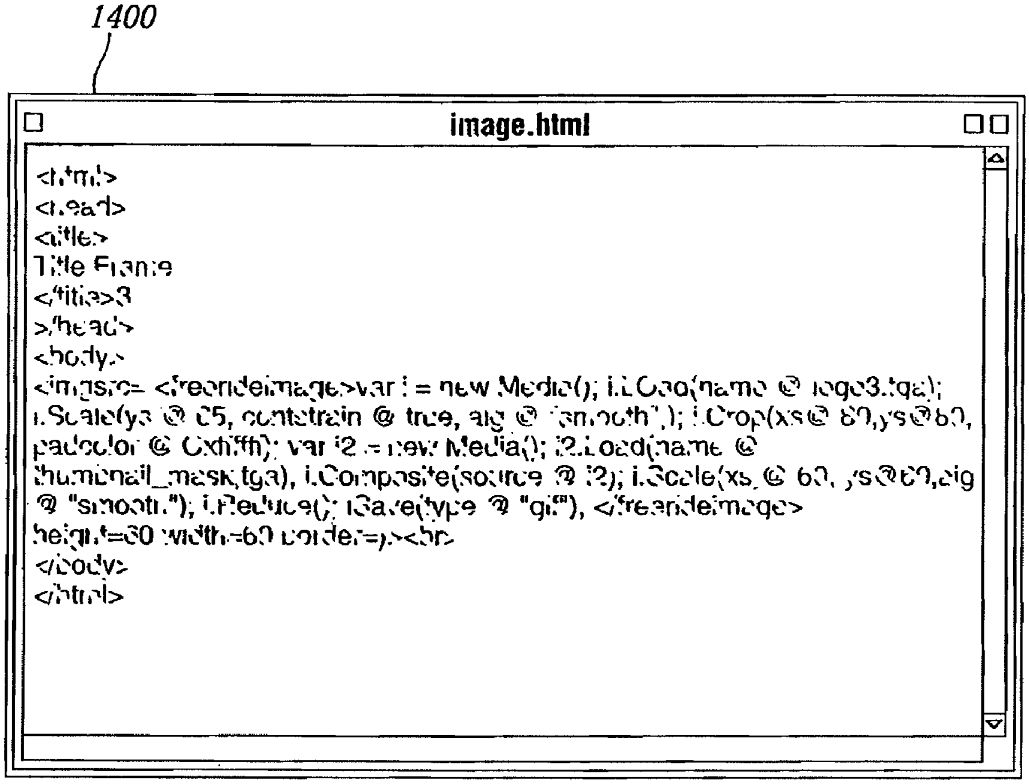


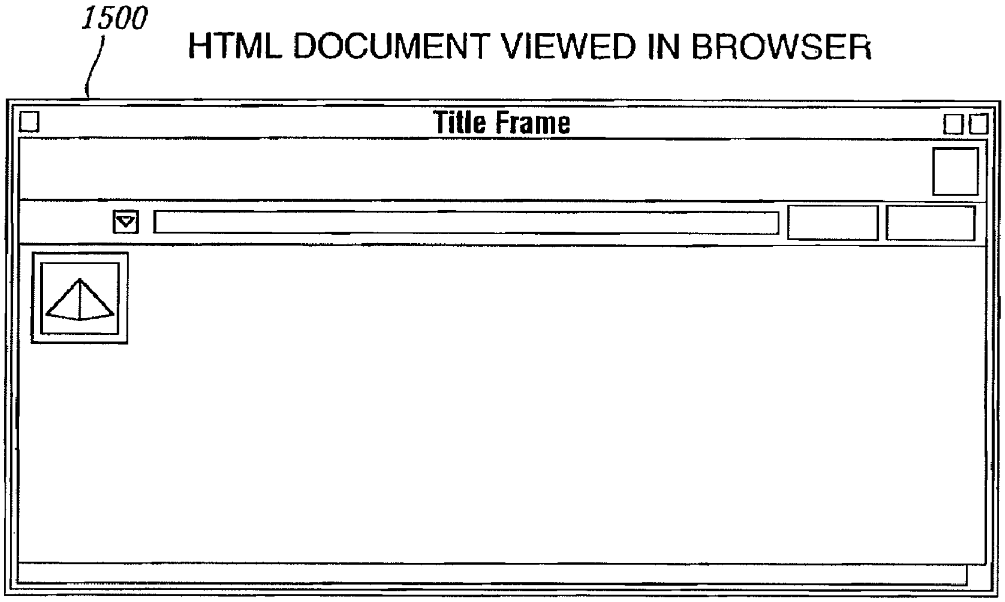
FIG.13

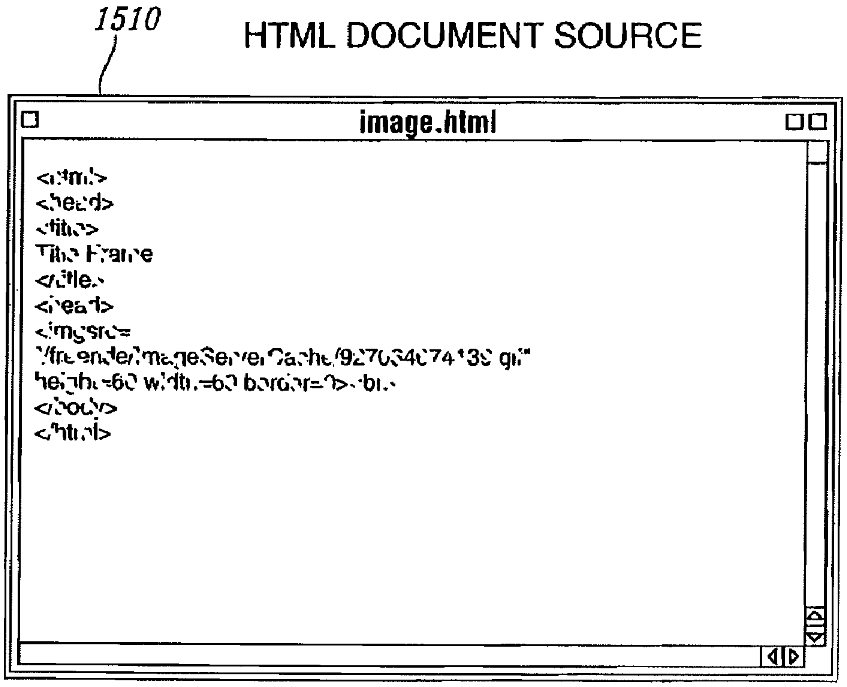
HTML DOCUMENT WITH PROPRIETARY TAG



```
<html>
<head>
<title>
Title Frame
</title>3
>/head>
<body>
<imgsrc= <freerideimage>var i = new Media(); i.Load(name @ logo3.tga);
i.Scale(ys @ 65, constrain @ true, alg @ "smooth",); i.Crop(xs@ 60,ys@60,
padcolor @ 0xffffff); var i2 = new Media(); i2.Load(name @
thumbnail_mask.tga), i.Composite(source @ i2); i.Scale(xs @ 60, ys@60,alg
@ "smooth"); i.Reduce(); i.Save(type @ "gif"), </freerideimage>
height=60 width=60 border=0><br>
</body>
</html>
```

FIG.14





```
<html>
<head>
<title>
Title Frame
</title>
</head>
<imgsrc=
"/frerender/ImageServerCache/927034074139.gif"
height=60 width=60 border=0><br>
</body>
</html>
```

FIG. 15

GENERATED GIF IMAGE

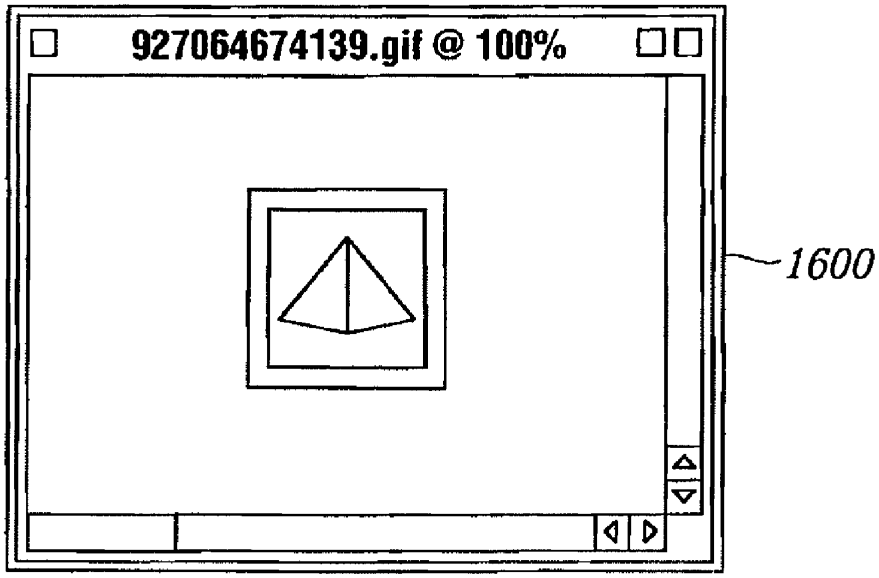


FIG.16

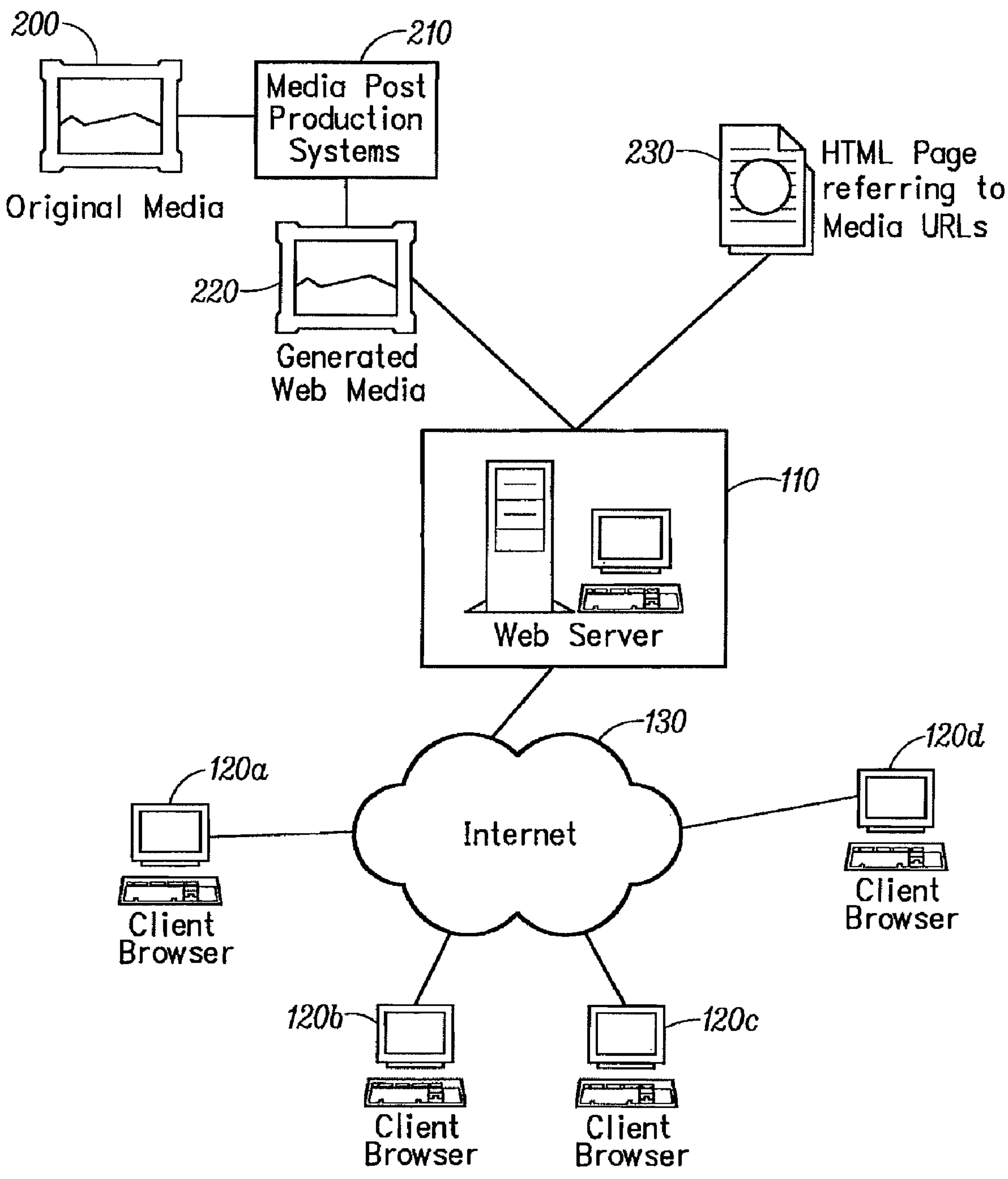

200
Original Media
210
Media Post Production Systems
220
Generated Web Media
230
HTML Page referring to Media URLs
110
Web Server
130
Internet
120a
Client Browser
120b
Client Browser
120c
Client Browser
120d
Client Browser

FIG. 17

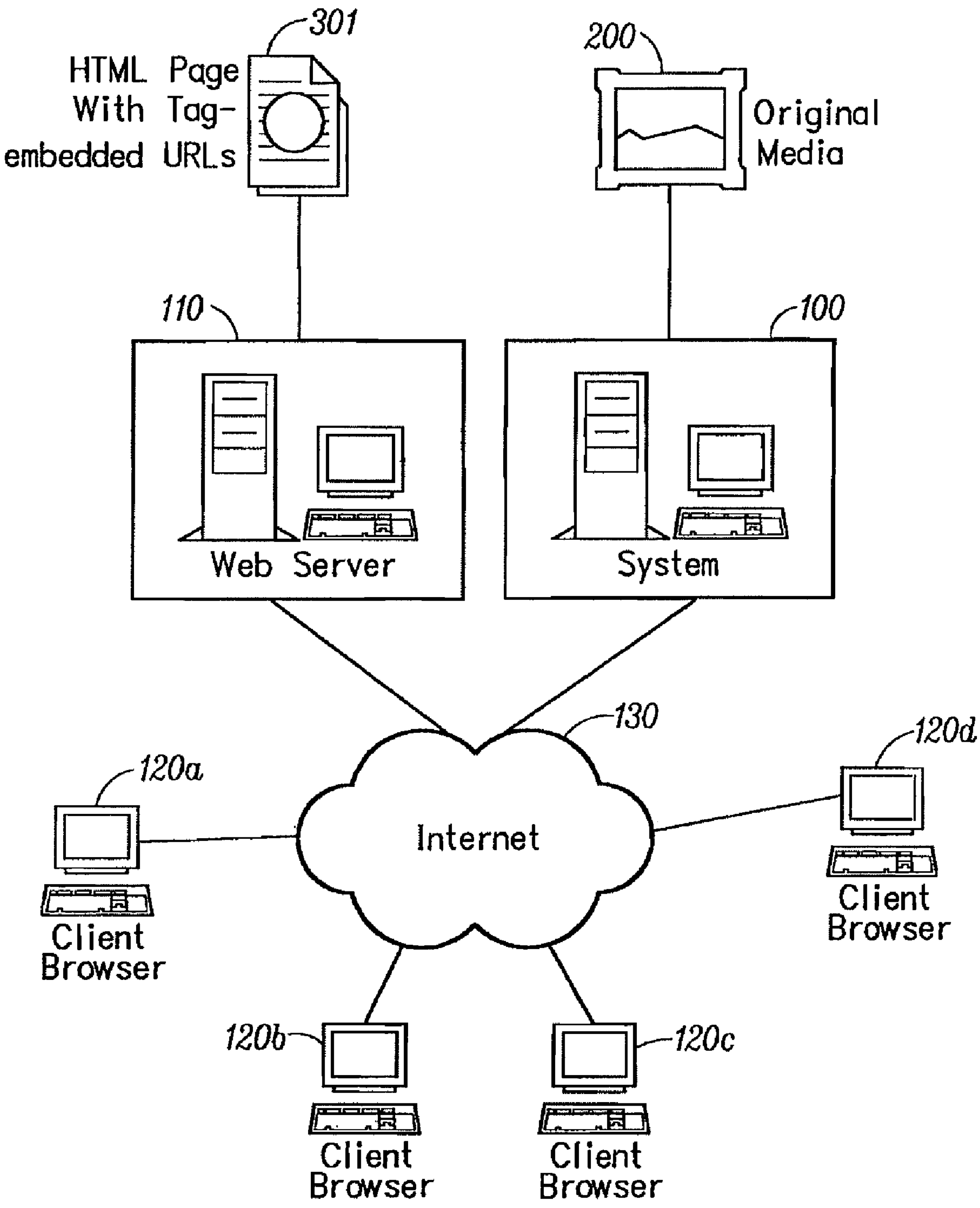

301
HTML Page With Tag-embedded URLs
200
Original Media
110
Web Server
100
System
130
Internet
120a
Client Browser
120b
Client Browser
120c
Client Browser
120d
Client Browser

FIG. 18

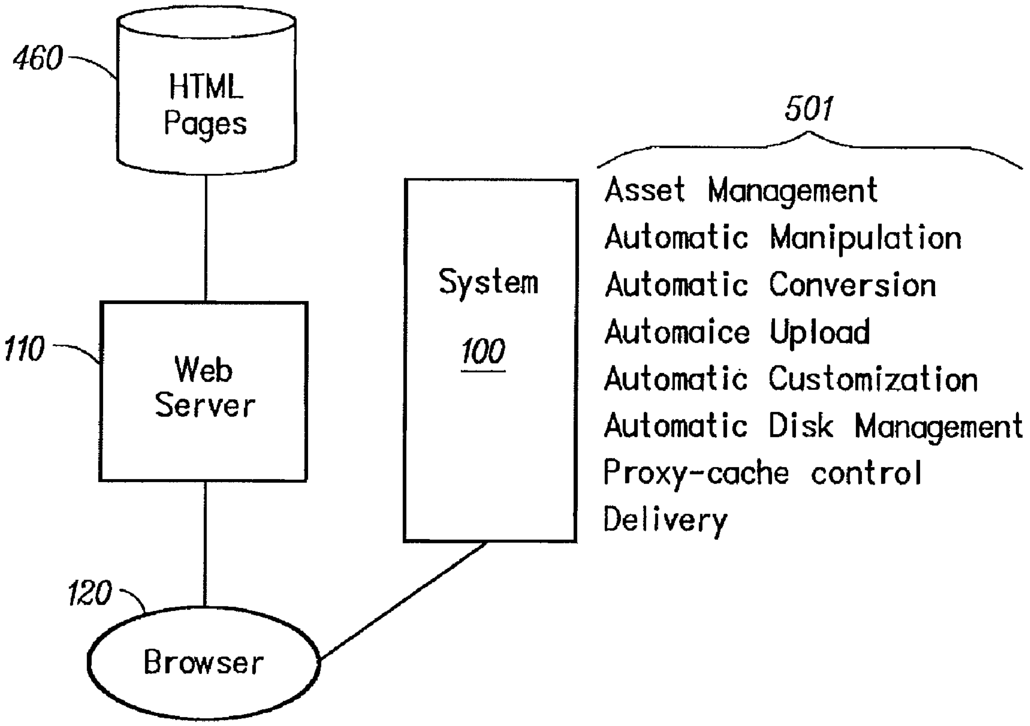

460
HTML Pages
110
Web Server
120
Browser
System
100
501
Asset Management
Automatic Manipulation
Automatic Conversion
Automaice Upload
Automatic Customization
Automatic Disk Management
Proxy-cache control
Delivery

FIG. 19

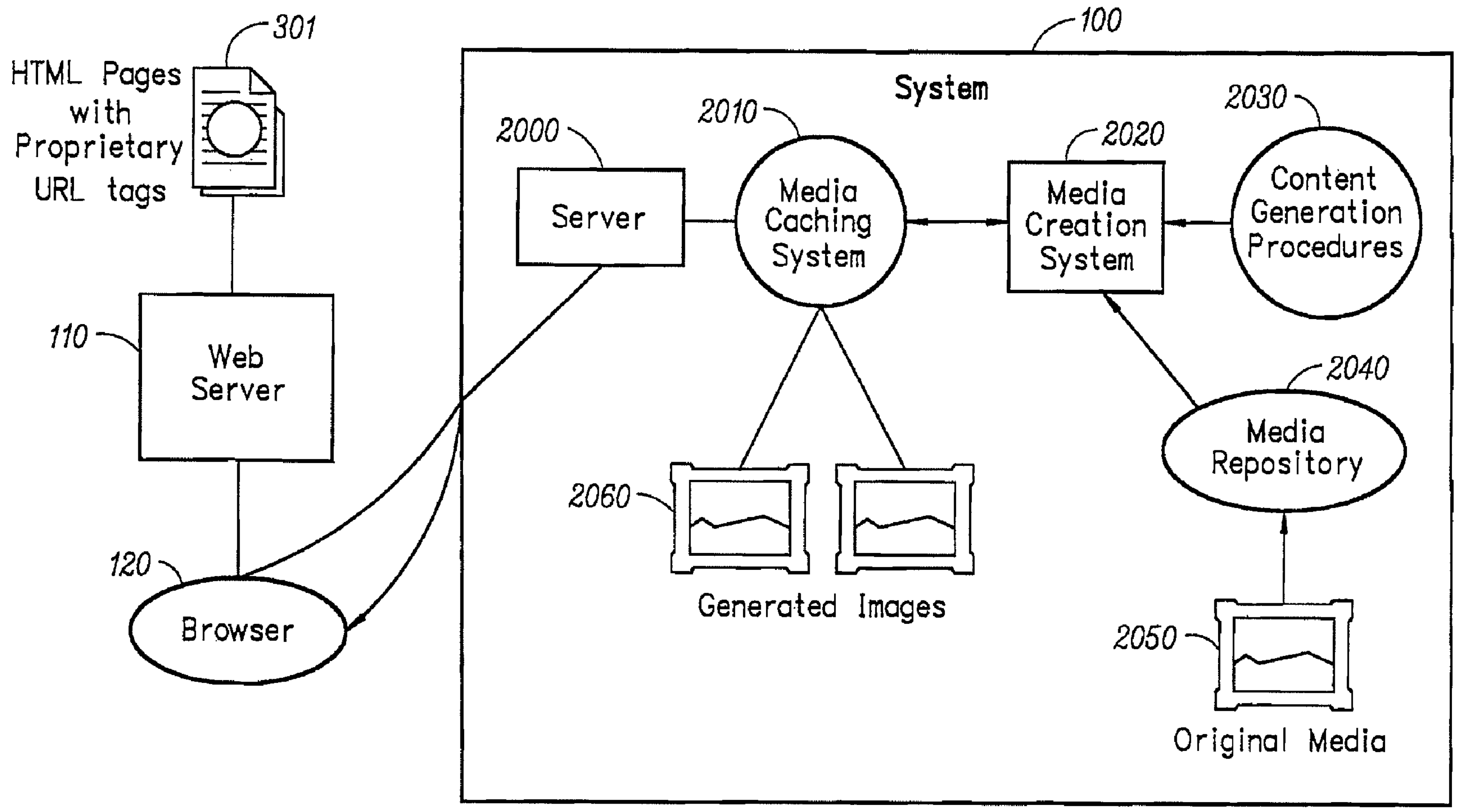

301
HTML Pages with Proprietary URL tags
110
Web Server
120
Browser
100
System
2000
Server
2010
Media Caching System
2020
Media Creation System
2030
Content Generation Procedures
2040
Media Repository
2050
Original Media
2060
Generated Images

FIG. 20

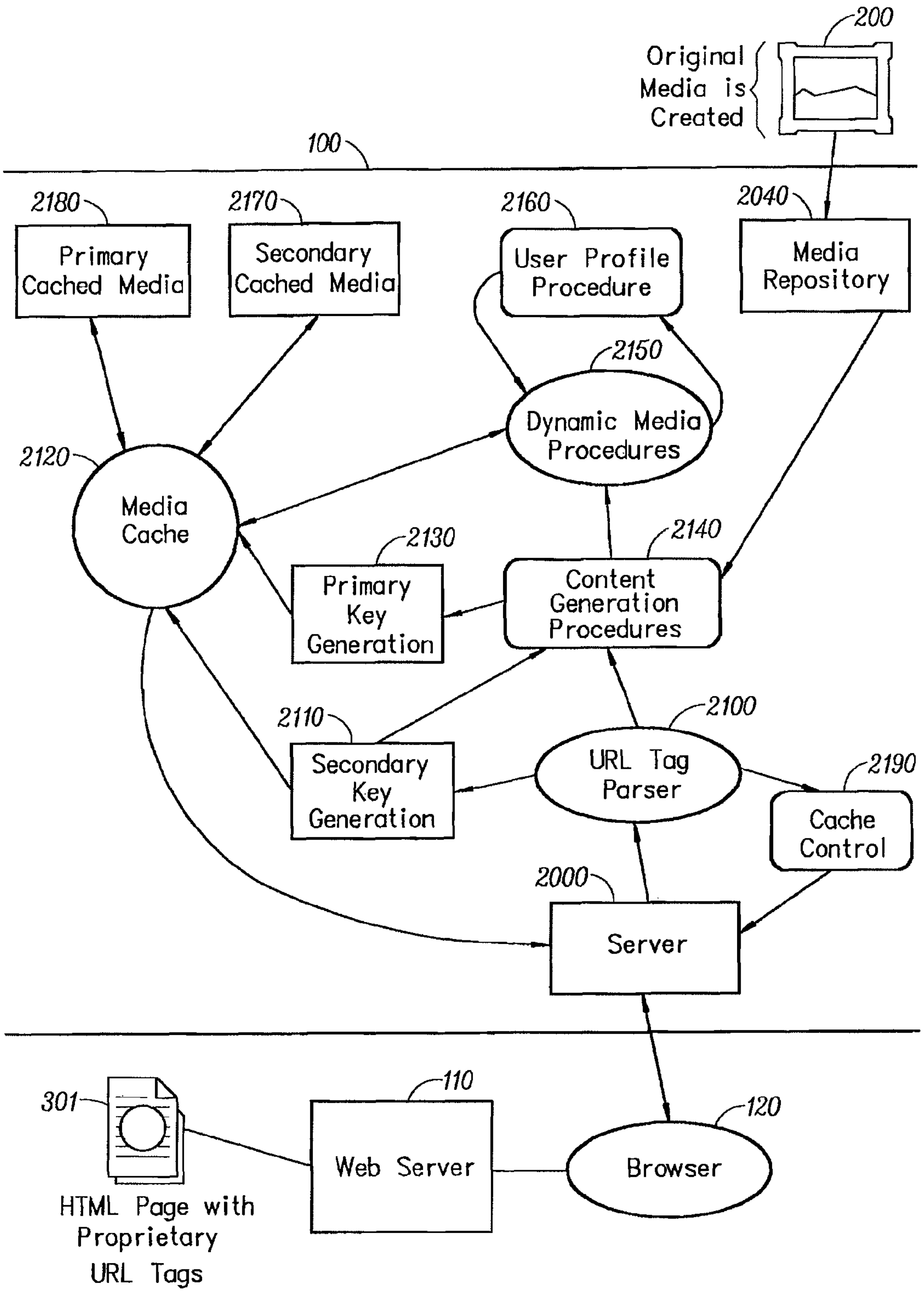

200
Original Media is Created
100
2180
Primary Cached Media
2170
Secondary Cached Media
2160
User Profile Procedure
2040
Media Repository
2150
Dynamic Media Procedures
2120
Media Cache
2130
Primary Key Generation
2140
Content Generation Procedures
2110
Secondary Key Generation
2100
URL Tag Parser
2190
Cache Control
2000
Server
301
HTML Page with Proprietary URL Tags
110
Web Server
120
Browser

FIG. 21

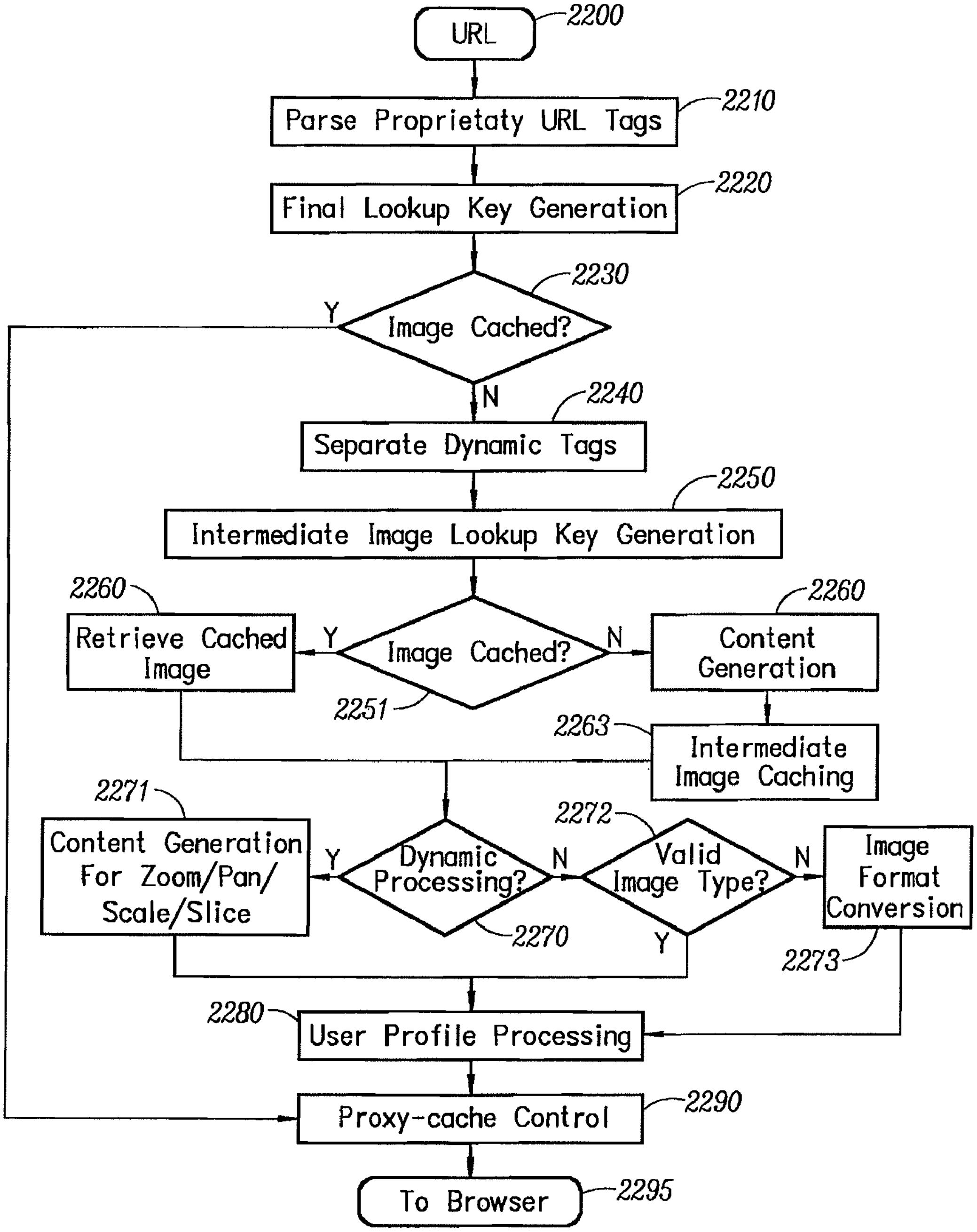

URL
2200
Parse Proprietaty URL Tags
2210
Final Lookup Key Generation
2220
Image Cached?
2230
Y
N
Separate Dynamic Tags
2240
Intermediate Image Lookup Key Generation
2250
2260
Retrieve Cached Image
Y
Image Cached?
N
Content Generation
2260
2251
2263
Intermediate Image Caching
2271
Content Generation For Zoom/Pan/ Scale/Slice
Y
Dynamic Processing?
N
2272
Valid Image Type?
N
Image Format Conversion
2270
Y
2273
2280
User Profile Processing
Proxy-cache Control
2290
To Browser
2295

FIG. 22

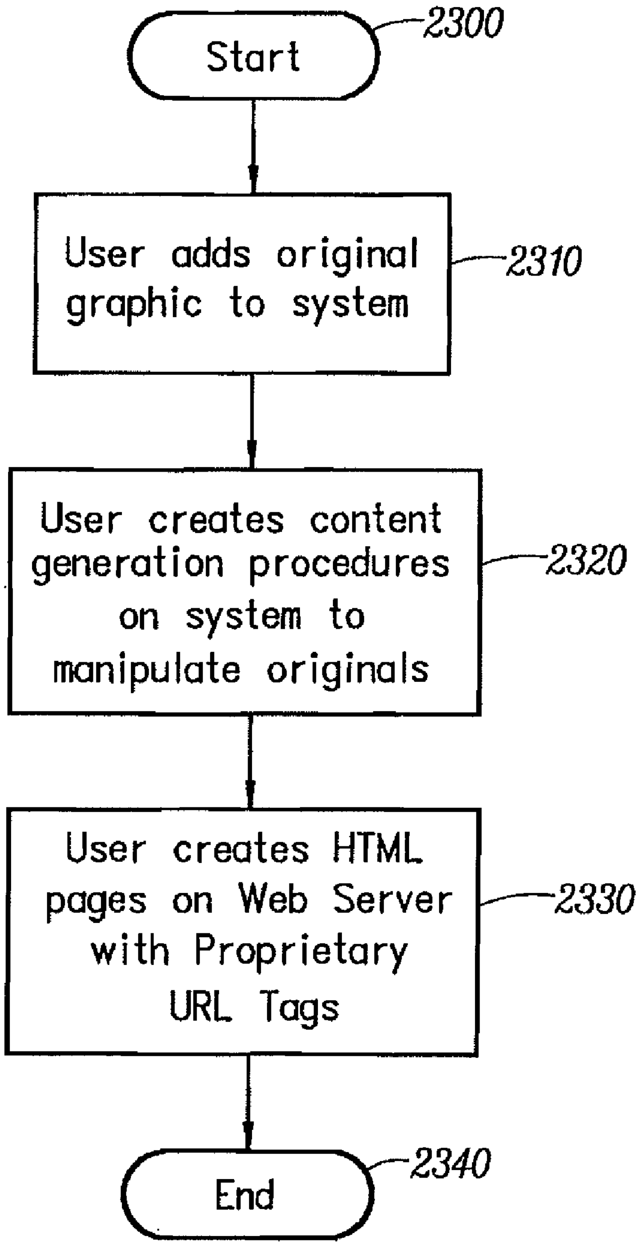

Start
2300
User adds original graphic to system
2310
User creates content generation procedures on system to manipulate originals
2320
User creates HTML pages on Web Server with Proprietary URL Tags
2330
End
2340

FIG. 23

AUTOMATED MEDIA DELIVERY SYSTEM

CROSS REFERENCE TO RELATED APPLICATIONS

This application is a Divisional of U.S. Ser. No. 12/173,747, filed Jul. 15, 2008, which is a Divisional of U.S. Ser. No. 11/269,916, filed Nov. 7, 2005 now abandoned, which is a Continuation-in-Part of U.S. Ser. No. 09/929,904, filed Aug. 14, 2001, now U.S. Pat. No. 6,964,009 granted on Nov. 8, 2005, which is a Continuation of U.S. Ser. No., 09/425,326, filed Oct. 21, 1999, now U.S. Pat. No. 6,792,575, granted on Sep. 14, 2004, each of which is hereby incorporated in its entirety by this reference thereto.

BACKGROUND OF THE INVENTION

1. Technical Field

The invention relates to software systems. More particularly, the invention relates to an Internet server-based software system that provides delivery of automated graphics and other media to Web sites for access by an end user or consumer.

2. Description of the Prior Art

Most Web sites today are primarily handmade. From the guy publishing a simple online technology newsletter from his home, to the Fortune 1000 company's multi-tiered site with hundreds of pages of text, images, and animations, the Web developer and each of his HTML-coding and graphics-producing coworkers toil page by page and image by image. Thousands of established online companies employ hundreds of highly-skilled workers just to produce and maintain their Web sites. After all, the Web is now a major selling vehicle and marketing medium for many of these companies. The Web has even sprouted service industries such as, for example, public companies with multi-billion dollar valuations created just to consult and produce Web sites for others.

Most Web developers who use established WYSIWYG tools in the industry still must produce each page on their Web site one by one. The same rate applies to preparing and placing images, animations, and other visual assets. Each page represents its own set of issues ranging from whether to use GIF, JPEG, or PNG file formats, to finding the optimum bit depth for each image to ensure the fastest downloading through the different browsers of the consumer. The bottlenecked state of the customer's workflow to produce graphics for Web pages can be described as follows:

Current Workflow for Creating Web Graphics
Original Artwork/Asset Creation
  Use third-party point products
Asset Editing
  Scale/reduce/slice
Asset Format Conversion
  JPEG/GIF/PNG
Asset Staging
  Place in Web file system
  Edit HTML
Create/Modify HTML for particular page
Store HTML on Web server
View final pages
Repeat process for each version of each graphic on each page
Estimated time
Two hours per page times the number of pages

Also, from a user's perspective, the current state of the art is to offer the consumer zooming and panning capabilities so that by clicking on an image the consumer can view more closely or from a different angle. On the horizon are pages with three-dimensional imagery that enable a user to move around a page that can look more like a room than a brochure. While interesting, these features are merely incremental improvements to a consumer's surfing experience.

D. C. A. Bulterman, *Models, Media, and Motion: Using the Web to Support Multimedia Documents*, Proceedings of 1997 International Conference on Multimedia Modeling, Singapore, 17-20 Nov. 1997 discloses "an effort underway by members of industry, research centers and user groups to define a standard document format that can be used in conjunction with time-based transport protocols over the Internet and intranets to support rich multimedia presentations. The paper outlines the goals of the W3C's Synchronized Multimedia working group and presents an initial description of the first version of the proposed multimedia document model and format."

Text and Graphics on UMI's ProQuest Direct: The Best (yet) of both Worlds, Online, vol. 21, no. 2, pp. 73-7, March-April 1997 discloses an information system that offers "periodical and newspaper content covering a wide range of business, news, and professional topics . . . letting the user search both text and graphics and build the product to suit. Articles can be retrieved in varying levels of detail: citation, abstracts, full text, and text with graphics. Images come in two flavors: Page Image, a virtual photocopy, and Text+Graphics, in which graphics are stored separately from the text and are manipulable as discrete items . . . . [The system] comes in two versions: Windows and Web."

John Mills Dudley, Network-Based Classified Information Systems, AU-A-53031/98 (Aug. 27, 1998) discloses a "system for automatically creating databases containing industry, service, product and subject classification data, contact data, geographic location data (CCG-data) and links to web pages from HTML, XML, or SGML encoded web pages posted on computer networks such as Internets or Intranets . . . . The . . . databases may be searched for references (URLs) to web pages by use of enquiries which reference one or more of the items of the CCG-data. Alternatively, enquiries referencing the CCG-data in the databases may supply contact data without web page references. Data duplication and coordination is reduced by including in the web page CCG-data display controls which are used by web browsers to format for display the same data that is used to automatically update the databases."

Cordell et al, Automatic Data Display Formatting with A Networking Application, U.S. Pat. No. 5,845,084 (Dec. 1, 1998) discloses a placeholder image mechanism. "When a data request is made, the data transfer rate is monitored. When the receive data transfer rate is slow, and the data contains an embedded graphical image of unknown dimensions, a small placeholder image is automatically displayed for the user instead of the actual data. The small placeholder image holds a place on a display device for the data or the embedded graphical image until the data or embedded graphical image is received. When embedded graphical image is received, the placeholder image is removed, and the display device is reformatted to display the embedded graphical image."

Jonathon R. T. Lewis, System For Substituting Tags For Non-Editable Data Sets In Hypertext Documents And Updating Web Files Containing Links Between Data Sets Corresponding To Changes Made To The Tags, U.S. Pat. No. 5,355,472 (Oct. 11, 1994) discloses a "hypertext data processing system wherein data sets participating in the hypertext document may be edited, the data processing system inserting tags into the data sets at locations corresponding to the hypertext links to create a file which is editable by an editor and the data processing system removing the tags, generating a revised data set and updating the link information after the editing process. Its main purpose is to preserve the linking hierarchy that may get lost when the individual data sets get modified."

Wistendahl et al, System for Mapping Hot Spots in Media Content Interactive Digital Media Program, U.S. Pat. No. 5,708,845 (Jan. 13, 1998) discloses a "system for allowing media content to be used in an interactive digital media (IDM) program [that] has Frame Data for the media content and object mapping data (N Data) representing the frame addresses and display location coordinates for objects appearing in the media content. The N Data are maintained separately from the Frame Data for the media content, so that the media content can be kept intact without embedded codes and can be played back on any system. The IDM program has established linkages connecting the objects mapped by the N Data to other functions to be performed in conjunction with display of the media content. Selection of an object appearing in the media content with a pointer results in initiation of the interactive function. A broad base of existing non-interactive media content, such as movies, videos, advertising, and television programming can be converted to interactive digital media use. An authoring system for creating IDM programs has an object outlining tool and an object motion tracking tool for facilitating the generation of N Data. In a data storage disk, the Frame Data and the N Data are stored on separate sectors. In a network system, the object mapping data and IDM program are downloaded to a subscriber terminal and used in conjunction with presentation of the media content."

Rogers et al, Method for Fulfilling Requests of A Web Browser, U.S. Pat. No. 5,701,451 (Dec. 23, 1997) and Lagarde et al, Method for Distributed Task Fulfillment of Web Browser Requests, U.S. Pat. No. 5,710,918 (Jan. 20, 1998) disclose essentially "improvements which achieve a means for accepting Web client requests for information, obtaining data from one or more databases which may be located on multiple platforms at different physical locations on an Internet or on the Internet, processing that data into meaningful information, and presenting that information to the Web client in a text or graphics display at a location specified by the request."

Tyan et al, HTML Generator, European Patent Application No. EP 0843276 (May 20, 1998) discloses "generating an HTML file based on an input bitmap image, and is particularly directed to automatic generation of an HTML file, based on a scanned-in document image, with the HTML file in turn being used to generate a Web page that accurately reproduces the layout of the original input bitmap image."

TrueSpectra has a patent pending for the technology employed in its two products, IrisAccelerate and IrisTransactive. These products are designed for zooming and panning and simple image transformations and conversions, respectively. They support 10 file formats and allow developers to add new file formats via their SDK. They do not require the use of Flashpix for images. However, their documentation points out that performance is dependent on the Flashpix format. The system would be very slow if a non-Flashpix format was used.

TrueSpectra allows the image quality and compression to be set for JPEGs only. The compression setting is set on the server and all images are delivered at the same setting.

TrueSpectra has a simple caching mechanism. Images in the cache can be cleared out automatically at certain times and it does not have any dependency features for image propagation. The Web server needs to be brought down in order to update any original assets.

TrueSpectra does not require plug-ins to operate features such as zooming/panning or compositing. The alternative to plug-ins is using their Javascript or active server page technology. These technologies are used by many Web sites to provide interactivity, but not all Web browsers work correctly with these technologies. TrueSpectra relies on Flashpix as its native file format and does not support media types such as multi-GIFs and sound formats. Flashpix files are typically larger than most file formats. Access to files is faster for zooming and panning, but appears to be quite slow.

The key to IrisTransactive is the compositing subsystem. It requires three things to build a shopping solution using image composition.

1) The original images must be created. It is suggested that the image be converted to Flashpix for better performance.
2) All of the individual images must be described in XML using the image composer program. The program allows the editor to specify anchor points, layer attributes, and layer names. The resulting file is between 5k and 50k.
3) The Web designer must place HTML referring to the XML in the Web site. By specifying parameters to the XML, the Web designer can turn on or off layers.

The herein above process for compositing images enables Web designers to create shopping sites. However, a lot of overhead is the result. The XML documents add 5k-50k to a Web site. The compositing commands that are embedded in the HTML are difficult to understand. And, because the compositing feature requires several steps to implement, it is not suitable for every image on a Web site. The process seems to be designed for the specific purpose of shopping.

MediaBin™ is limited to activities behind the firewall automating only the "post-creative busywork." In addition, MediaBin requires the use of an application server to function through a web interface. Thus images may not be directly added to any existing web page.

Macromedia's Generator operates by embedding variables in their proprietary Flash format. Therefore the actual imaging operations are somewhat limited and cannot be controlled directly from a web page request.

MGI Software sells point solutions that require end-users to download a viewer to process a proprietary image format.

PictureIQ offers a server-side image-processing appliance that provides a limited set of Photoshop functionalities. This appliance runs on the web-page server, processes information embedded in the web page, and rewrites the web page with image data.

The disclosed prior art fail to provide systems and methodologies that result in a quantum leap in the speed with which they can modify and add images, video, and sound to sites, in the volume of data they can publish internally and externally, and in the quality of the output. The development of such an automated media delivery system would constitute a major technological advance.

It would be advantageous to empower an end user with flexibility and control by providing interactive page capabilities.

It would be advantageous from an end user's perspective to generate Web pages that contain active graphics. For example, clicking on a Corvette image will cause a simple menu to pop up suggesting alternative colors and sizes in which to see the car. Clicking on portions of the image, such as a fender, can call up a close-in view of the fender.

It would be advantageous to provide an automated graphics delivery system that becomes part of the Web site infrastructure and operates as part of the Web page transaction and that thereby provides a less expensive and less time-consuming process.

It would be advantageous to provide a system for automated processing and delivery of media (images, video, and sound) to a Web server whereby it eliminates the laborious post-production and conversion work that must be done before a media asset can be delivered on a Web server.

It would be advantageous to create a dynamic Web site, wherein images are generated on demand from original assets, wherein only the original assets need to be updated, and wherein updated changes propagate throughout the site.

It would be advantageous to provide a system that generates media based on current Web server traffic thereby optimizing throughput of the media through the Web server.

It would be advantageous to provide a system that generates media that is optimized for the Web client, wherein client connection speed determines optimum quality and file size.

It would be advantageous to provide a system that generates media, whereby the media is automatically uploaded.

It would be advantageous to provide a system that automatically caches generated media so identical requests can be handled without regeneration of images.

It would be advantageous to provide a system that resides behind the Web server, thereby eliminating security issues.

It would be advantageous to provide a system wherein the client browser does not require a plug-in.

It would be advantageous to provide a system wherein the system does not require any changes to a Web server.

It would be advantageous to provide a system wherein the system manages the Web server media cache.

It would be advantageous to provide a system wherein the Web media is generated only if requested by a client browser.

It would be advantageous for a system to reduce the need for a Web author to create different versions of a Web site, the system automatically handling image content.

It would be advantageous to provide dynamic imaging capabilities, have a more complete set of image processing functionality, and be controlled directly through an image URL.

It would be advantageous to provide an end-to-end solution requiring only a standard browser that is completely controllable using the proprietary tags contained within a simple image link in the web page.

It would be advantageous to run an image application as a separate server controlled directly by single image requests to that server, such that any web server, even one that is only sending static HTML can access imaging features.

SUMMARY OF THE INVENTION

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, generated media is cached such that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a schematic diagram showing the placement of the system within a current Web infrastructure according to the invention;

FIG. **2** is a schematic diagram showing how a typical Web site delivers an HTML document and its graphics to a Web browser according to the prior art;

FIG. **3** is a schematic diagram showing delivery of an HTML document and media to a Web browser according to the invention;

FIG. **4** is a schematic diagram showing the components involved in Web site administration according to the prior art;

FIG. **5** is a schematic diagram showing the components of the system involved in Web site administration according to the invention;

FIG. **6** is a simple overview showing the components of the system according to the invention;

FIG. **7** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to the invention;

FIG. **8** is a flow chart showing an authoring process according to the invention;

FIG. **9** is a flow chart showing an HTML parsing process according to the invention;

FIG. **10** is a flow chart showing a media creation process according to the invention;

FIG. **11** is a screen shot showing an administration tool according to the invention;

FIG. **12** displays a structure of a database record used for the system according to the invention;

FIG. **13** shows original media to be processed according to the invention;

FIG. **14** shows a portion on an HTML document with a proprietary tag according to the invention;

FIG. **15** shows an HTML document and an HTML document source according to the invention;

FIG. **16** shows a generated GIF image according to the invention;

FIG. **17** is a schematic diagram of an image system within a typical Web infrastructure according to the invention;

FIG. **18** is a schematic diagram showing delivery of an HTML document and original media according to the invention;

FIG. **19** is a schematic diagram showing components of Web site administration according to a preferred embodiment of the invention;

FIG. **20** is a simple overview showing components of the image system according to a preferred embodiment of the invention;

FIG. **21** is a schematic diagram showing process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention;

FIG. **22** shows a flowchart of a content generation procedure according to a preferred embodiment of the invention; and

FIG. **23** is a flow chart showing an authoring process according to a preferred embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, the generated media is cached so that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

A detailed description of such automatic media delivery system operating in parallel with existing Web site infrastructure is found below in the section under the heading as such.

FIG. **1** is a schematic diagram showing the placement of the system within a current Web infrastructure according to a preferred embodiment of the invention. The system **100** is attached to a Web server **110**, which is connected to multiple client browsers **120**(*a-d*) via the Internet **130**.

FIG. **2** is a schematic diagram showing how a typical Web site delivers an HTML document and its graphics to a Web browser according to the prior art. An original media **200** is passed to post-production systems **210**, wherein the media **200** is manipulated by hand and prepared for the Web. The result is a Web media **220**. The Web media **220** and an associated HTML document **230** referring to the media **220** by media tags are input to a Web server **110** for a Web browser **120** to view via the Internet **130**.

FIG. **3** is a schematic diagram showing delivery of an HTML document and media to a Web browser according to a preferred embodiment of the invention. An original media **200** and an HTML document embedded with proprietary media tags **300** are input into the system **100**. The system **100** generates a Web-safe media **220** and a modified HTML document **230** that refers to the Web media, and automatically loads them onto the Web server **110** for view by a Web browser **120** via the Internet **160**.

FIG. **4** is a schematic diagram showing components involved in Web site administration according to the prior art. Original media assets **400** are original images, video, or sound that have not been prepared for the Web. Web sites usually need to manage the placement of media on the network for easy retrieval by Web designers. Post-production systems **410** vary from Web site to Web site. Post-production systems **410** are usually custom procedures that Web designers use to convert an original media, such as an image, to one that can be displayed on the Web. Post-production systems **410** also upload finished images to Web image systems. Web images **420** are Web versions of the original images. Web images **420** are ready for retrieval by the Web server **110** to be delivered to a Web browser **120**. Any image to be modified or updated must pass through the herein above three components before it can be delivered to the Web browser **120**. HTML pages **460** have references to Web images **420**.

FIG. **5** is a schematic diagram showing the components involved in Web site administration according to a preferred embodiment of the invention. Web site administration is simplified using the claimed invention. Asset management, automatic image manipulation, automatic image conversion, automatic image upload, and automatic disk management **500** are provided by the claimed invention.

FIG. **6** is a simple overview showing the components of the system according to a preferred embodiment of the invention. HTML with proprietary tags **300** is the original HTML document that is embedded with proprietary tags which describe how the images are to be manipulated for the Web. Java servlet engine **600** is a third-party product that allows the system **100** to interface with the Web server **110** and execute Java servlet code. The Web server **110** is third-party software that delivers Web pages to a Browser **120**. The Browser **120** views Web pages that are sent from the Web server **110**. Modified HTML with system created images **230** are a final result of the system. Modified HTML **230** is a standard HTML document without proprietary embedded tags and with standard Web graphics.

The System.

A preferred embodiment of the system **100** is provided.

HTML parsing subsystem **610** parses through an HTML document and searches for proprietary tags. If it finds a proprietary tag it hands it to a media caching subsystem **620** for further processing. The media caching subsystem **620** returns a standard HTML tag. The HTML parsing subsystem **610** then replaces the proprietary tag it found with the returned tag. The parsing subsystem **610** then continues searching for a next proprietary tag, repeating the process herein above. The process is finished when no more proprietary tags can be found.

The media caching subsystem **620** determines if an image has been created for the requested proprietary tag. If the image has already been created and the files that built that image have not been modified, the media caching subsystem **620** returns an HTML tag that refers to a previously-generated image. If the image has not been created, the media caching subsystem **620** hands the HTML tag to a media creation subsystem **630**. The media creation subsystem **630** returns an image to the media caching subsystem **620**. The media caching subsystem **620** adds the created image and the HTML tag to a media cache database **640**.

The media cache database **640** contains references to the created images **645**. In a preferred embodiment, the references are the script used to create the image, the names of the images used to create the image, the dates of those files, and the HTML that represents the created image. The media caching subsystem **620** performs lookups in this database to determine if the image has been created. If the image has not been created the media caching subsystem **620** calls upon the media creation subsystem **630** to create the image and then store the results in the media cache database **640**.

The media creation subsystem **630** takes a proprietary tag from the media caching subsystem **620** and generates an image. The image is generated by deciphering the tag and handing it to the media processing engine **650**. After the image is created, the media creation subsystem returns the name of the newly created image to the media caching subsystem **620**.

The media processing engine **650** interprets the proprietary tag and generates the image. The media processing engine **650** looks up images in a media repository to obtain the location of the original file.

The media repository **660** contains original images **665** used in the system **100**.

FIG. **7** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention. An original media **200** is created. The media **200** is placed into the system **100** in the media repository **660**. Similarly, an HTML document with proprietary tags **300** is created and placed on a Web server **110**. A user requests a Web page from a Web browser **120**. The Web server **110** passes the requested page to an HTML parser **610**. The HTML parser **610** parses HTML looking for media tags. The parser **610** looks up media tags in a media tags database **640**. If the media tag is found, then the system **100** produces a modified HTML document **230**. Otherwise, the media creation subsystem **630** uses the media tag to generate a Web media **220**. The generated Web media **220** is placed in a media cache subsystem **620**. The proprietary media tag is converted by a converter **700** to a standard HTML tag that refers to the generated media **220** in cache. The media tag and the HTML equivalent are stored in the media tags database **640**. Media tags are replaced by standard HTML equivalent to provide a modified HTML document **230**. The modified HTML document **230** is delivered to the Web server **110**. The Web server **100** delivers the modified HTML document **230** to the browser **120** via the Internet for a user to view.

FIG. **8** is a flow chart showing an authoring process according to a preferred embodiment of the invention. The process starts (**800**) when a user adds an original graphic to the system (**810**). The user then creates an HTML document that contains proprietary media tags (**820**). The user then places the HTML document on a Web server (**830**) and ends the authoring process (**840**).

FIG. **9** is a flow chart showing an HTML parsing process according to a preferred embodiment of the invention. The process starts (**900**) when a consumer requests a Web page (**910**). A Web server hands the request of the Web page to the system (**920**). The system parses the Web page (**930**). The system looks for a media tag (**940**). If found, the system retrieves the HTML equivalent of the media tag (**950**) and replaces the media tag with the HTML equivalent tag (**960**). The system continues parsing the Web page for tags (**970**) by returning to step (**940**). When no more tags are found, the system delivers the modified Web page to the Web server (**980**) and therein ends the process (**990**).

FIG. **10** is a flow chart showing a media creation process according to a preferred embodiment of the invention. The process starts (**1000**) when the system requests an HTML equivalent to a proprietary media tag (**1010**). The Media tag is combined with bandwidth information (**1020**). The subsystem checks if the media tag already exists in the media tag database (**1030**). If it does, the subsystem checks if any of the original assets used to create the media have been changed (**1040**). If not, then the subsystem retrieves the HTML equivalent tag from the database (**1050**) and returns the HTML equivalent tag to the requesting system (**1060**). If any of the original assets used to create the media have been changed (**1040**), then the subsystem removes the media tag entry from the media database (**1070**) and creates the media using the media tag (**1080**). The subsystem then stores the media in a media cache (**1090**). The subsystem generates the HTML referring to the generated media (**1100**) and places the media tag and the HTML equivalent in the media tag database (**1110**). The HTML equivalent is returned to the requesting system (**1060**) and the process stops (**1120**).

The differences between using HTML and the proprietary tags disclosed herein are noted. HTML allows Web designers to create Web page layouts. HTML offers some control of the images. HTML allows the Web designer to set the height and width of an image. However, all of the other image operations disclosed herein are supported by the claimed invention and are not supported by HTML.

Table A herein below provides the claimed proprietary tags according to a preferred embodiment of the invention. The use of the term “freeride” refers to an internal code name for the invention.

TABLE A

Tags
Generate image
<freerideimage> mediascript </freerideimage>
Generate a standard Web image.
Generate thumbnail image linked to full image
<freerideimagethumbnail> mediascript <xs=size ys=size /freerideimagethumbnail>
Generate a thumbnail of specified size and link it to the full size version.
Generate zoom and pan image
<freerideimagezoom> mediascript </freerideimagezoom>
Generate a zoomable/panable image.
Security
<freerideimagesecure> </freerideimagesecure>
Specifies that all images found between these tags are secured images and the system will determine access before generating.

Table B herein below provides the claimed script commands according to a preferred embodiment of the invention. Additional commands may be added as needed.

TABLE B

Media processing script commands
Add Noise
Noise__AddNoise([amount=<value 1..999>] [gaussian] [grayscale])
This command adds noise to the image.
Adjust HSB
AdjustHsb([hue @ <value ±255>] [saturation @ <value ±255>] [brightness @ <value ±255>])
This command allows the HSB of an image to be altered. This can be applied to images of all supported bit-depths.
Adjust RGB
AdjustRgb([brightness @ <value ±255>] [contrast @ <value ±255>] [red @ <value ±255>] [green @ <value ±255>] [blue @ <value ±255>] [noclip @ <true, false>] [invert @ <true, false>])
This command allows the contrast, brightness, and color balance of an image to be altered.
Blur
Blur(radius @ <value 0..30>)
This command applies a simple blur filter on the image.

TABLE B-continued

Media processing script commands
Blur Convolve
Blur_Blur( )
This command commands perform a simple 3 × 3 convolution for blurring.
Blur Convolve More
Blur_MoreBlur( )
This command commands perform a stronger 3 × 3 convolution for blurring.
Blur Gaussian
Blur_GaussianBlur([radius=<value 0.1..250>])
This command applies a Gaussian blur to the image.
Blur Motion
Blur_MotionBlur([distance=<value 1..250>] [angle=<degrees>])
This command applies motion blurring to the image using the specified distance and angle.
Brush Composite
Composite(source @ {<User-Defined Media Object name>} [x @ <pixel>] [y @ <pixel>] [onto] [opacity @ <value 0..255>] [color @ <color in hexadecimal>] [colorize @ <true, false>] [saturation @ <value 0..255>])
This command composites the specified "brush" (foreground) image onto the current "target" (background) image.
Colorize
Colorize(color @ <color in hexadecimal> [saturation @ <value 0..255>])
This command changes the hue of the pixels in the image to the specified color.
Convert
Convert(rtype @ <bit-depth> {dither @ <value 0..10>])
This command converts the image to the specified type/bit-depth.
Convolve
Convolve(Filter @ <filtername>)
This command applies a basic convolution filter to the image. In a user interface driven system, the filters could be stored in files and edited/created by the user.
Crop/Resize Canvas
Crop([xs @ {<pixels>, percentage + "%">}] [ys @ {<pixels>, percentage + "%">}] [xo @ <left pixel>] [yo @ <top pixel>] [padcolor @ <color in hexadecimal>] [padindex @ <value 0..255>])
This command crops the media to a specified size.
Discard
Discard( )
This command removes the designated Media Object from memory.
Drop Shadow
DropShadow([dx @ <pixels>] [dy @ <pixels>] [color @ <color in hexadecimal>] [opacity @ <value 0..255>] [blur @ <value 0..30>] [enlarge @ <true, false>])
This command adds a drop shadow to the image based on its alpha channel.
Equal
Equal(source @ {<User-Defined Media Object name>})
This command compares the current media with the one specified. If the media are different in any way, an error value is returned.
Equalize
Equalize([brightness @ <−1, 0..20>] [saturation @ <−1, 0..20>])
This command equalizes the relevant components of the media. Equalization takes the used range of a component and expands it to fill the available range.
Export Channel
ExportGun(Channel @ <channelname>)
This command exports a single channel of the source as a grayscale image.
Find Edges
Stylize_FindEdges([threshold=<value 0..255>] [grayscale] [mono] [invert])
This command finds the edges of the image based on the specified threshold value.
Fix Alpha
FixAlpha( )
This command adjusts the RGB components of an image relative to its alpha channel.
Flip
Flip(<horizontal, vertical> @ <true, false>)
This command flips the media vertically or horizontally.
Frame Add
FrameAdd(Source @ <filename>)
This command adds the given frame(s) to the specified Media Object.
Glow/Halo
Glow(Size @ <value 0..30> [halo @ <value 0..size>] [color @ <color in hexadecimal>] [opacity @ <value 0..255>] [blur @ <value 0..30>] [enlarge @ <true, false>])
This command produces a glow or halo around the image based on the image's alpha.
High Pass
Other_HighPass([radius=<value 0.1..250>])
This command replaces each pixel with the difference between the original pixel and a Gaussian blurred version of the image.
Import Channel
ImportGun(channel @ <channel name> source @ {<User-Defined Media Object name>} [rtype @ <bit-depth>])
This command imports the specified source image (treated as a grayscale) and replaces the selected channel in the original.
Load

TABLE B-continued

Media processing script commands
Load(Name @ <filename> [type @ <typename>] [transform @ <true, false>]) This command loads a media from the specified file.
Maximum Other__Maximum([radius=<value 1..10>]) This command scans the area specified by the radius surrounding each pixel, and then replaces the pixel with the brightest pixel found.
Minimum Other__Minimum([radius=<value 1..10>]) This command scans the area specified by the radius surrounding each pixel, and then replaces the pixel with the darkest pixel found.
Normalize Normalize([clip @ <value 0..20>]) This command expands the volume of the sample to the maximum possible.
Pixellate Mosaic Pixellate__Mosaic([size=<value 2..64>]) This command converts the image to squares of the specified size, where each square contains the average color for that part of the image.
Pixellate Fragment Pixellate__Fragment([radius=<value 1..16>]) This command produces four copies of the image displaced in each direction (up, down, left, right) by the specified radius distance and then averages them together.
Quad Warp QuadWarp([tlx=<position>] [tly=<position>] [trx=<position>] [try=<position>] [blx=<position>] [bly=<position>] [brx=<position>] [bry=<position>] [smooth]) This command takes the corners of the source image and moves them to the specified locations, producing a warped effect on the image.
Reduce to Palette Reduce([colors @ <num colors>] [netscape @ <true, false>] [b&w @ <true, false>] [dither @ <value 0..10>] [dithertop @ <value 0..10>] [notbackcolor] [pad @ <true, false>]) This command applies a specified or generated palette to the image.
Rotate Rotate(Angle @ <value 0..359> [smooth @ <true, false>] [enlarge @ <true, false>] [xs @ <pixels>] [ys @ <pixels>]) This command rotates the media by the specified angle in degrees.
Rotate 3D Rotate3d([anglex @ <angle ±89>] [angley @ <angle ±89>] [distance @ <value>]) This command rotates the image in 3D about either the x-axis or y-axis.
Save Save([type @ <image-type>]) This command saves a media to the specified file.
Scale Scale([xs @ {<pixels>, <percentage + "%">}] [ys @ {<pixels>, <percentage + "%">}] [constrain @ <true, false>] [alg @ {"fast", "smooth", "outline"}] [x1 @ <pixels>] [y1 @ <pixels>] [x2 @ <pixels>] [y2 @ <pixels>]) This command scales the image to the specified size.
Select Selection([source @ <User-Defined media Object>}] [remove @ <true, false>] [invert @ <true, false>] [backcolor] [color=<color>] [index=<value>] [opacity @ <value 0..255>]) This command manages the selected region for the current Media Object.
Set Color SetColor([backcolor @ <color in hexadecimal>] [forecolor @ <color in hexadecimal>] [backindex @ <value 0..255>] [foreindex @ <value 0..255>] [transparency @ ("on","off")]) This command allows the background color, foreground color, and transparency state of an image to be set.
Set Resolution SetResolution([dpi @ <value>] [xdpi @ <value>] [ydpi @ <value>]) This command changes the DPI of the image in memory.
Sharpen Sharpen__Sharpen( ) This command sharpens the image by enhancing the high-frequency component of the image.
Sharpen More Sharpen__SharpenMore( ) This command sharpens the image by enhancing the high-frequency component of the image, but is stronger than the standard sharpening.
Stylize Diffuse Stylize__Diffuse([radius=<value 0..>] [lighten] [darken]) This command diffuses the image by randomizing the pixels within a given pixel radius.
Stylize Embose Stylize__Emboss([height=<value 1..10>] [angle=<degrees>] [amount=<percentage 1..500>]) This command converts the image to an embossed version.
Text Drawing DrawText(Text @ <string> Font @ <font file> [size @ <value>] [color @ <color in hexadecimal>] [smooth @ <true, false>] [<left, right, top, bottom> @ <true, false>] [x @ <pixel>] [y @ <pixel>] [wrap @ <pixel-width>] [justify @ {left,center,right}] [angle @ <angle>])

TABLE B-continued

Media processing script commands
This command composites the specified text string onto the image.
Text Making
MakeText(text @ <string> font @ <font file> [path @ <path to font directory>] [size @ <value 1..4095>]
[color @ <color in hexadecimal>] [smooth @ <true, false>] [wrap @ <pixel-width>]
[justify @ {left,center,right}] [angle @ <angle>])
This command creates a new image that includes only the specified text.
Trace Contour
Stylize__TraceContour([level=<value 0..255>] [upper] [invert])
This command traces the contour of the image at the specified level (for each gun).
Unsharpen Mask
Sharpen__UnsharpMask([amount=<percentage 1..500] [radius=<value 0.1..250>]
[threshold=<value 0..255>])
This command enhances the edges and detail of an image by exaggerating differences between the image and a gaussian blurred version of the same image.
Zoom
Zoom([xs @ <pixels>] [ys @ <pixels>] [scale @ <value>] [x @ <left pixel>] [y @ <top pixel>])
This command zooms in on a specified portion of the media and fits it to the specified size. This constitutes a crop followed by a scale.

Table C herein below provides a list of features provided by a preferred embodiment of the invention. It is noted that the list of features included in Table C is by no means complete. In other embodiments, the list of features is expanded or reduced as needed.

TABLE C

System Feature List
Reads and writes various file formats:
BMP, GIF, JPG, PNG, TIF, PICT, TGA, FSD, FPX;
Supports many image processing operations;
Dynamically creates Web images from original assets;
Dynamically creates thumbnail images;
Dynamically creates images that can be panned and zoomed without browser plug-ins or special file formats;
Automatically propagates changes of original assets throughout a Web site;
Uses an intelligent caching mechanism:
Clean up image cache on demand;
Eliminates orphaned image files; and
Optimizes Web server cache by providing most recent images;
Renders TrueType fonts on the server instead of browser;
Uses intelligent scaling of line drawings;
Allows Web designers to manipulate images with proprietary tags;
Preserves original image assets;
Optimizes Web server traffic by adjusting the bandwidth of graphics;
Optimizes images for client connection speed;
Allows clients to specify the quality of images on a Web site; and
Allows Web designers to dynamically create images by manipulating proprietary tags in their applications (server or client side).

FIG. **11** is a screen shot showing an administration tool according to a preferred embodiment of the invention. Specifically, FIG. **11** shows an administration page that contains cached images of generated scripts. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **12** displays a structure of a database record used for the system according to a preferred embodiment of the invention. A Script Table **1200** has 5 columns, Media Script **1210**, HTML Equivalent **1220**, Bandwidth **1230**, Generated File **1240**, and Dependency List **1250**. A Dependency Table **1260** has two columns, File Name **1270** and Modification Date **1280**.

Snowboard Store Example.

Background.

The snowboard store highlights several features of the claimed system. The snowboard store is an imaginary store that allows a user to configure his or her snowboard. The store consists of five logos, five board colors, and four boards. The consumer clicks on the buttons to change the snowboard represented in the middle of the screen. When the consumer has configured the snowboard they the snowboard can be purchased by selecting a buy button.

Prior Art Method.

To create the snowboard site today, the Web designer must render all possible combinations of the board. The number of combinations is five logos×five board colors×four boards=100. The designer also must render all the buttons. The creation process is very tedious and involves a lot of production work. Typically, most Web sites do not even attempt such an endeavor. Also, other issues must be addressed, such as, for example, updating the Web site and scripting. For example, updating a single logo involves updating a minimum of 20 images.

The prior art method sustains a graphic intensive site that requires management of at least 100 images. Updates to the Web site are time-consuming and prone to human error.

The Claimed Method.

A preferred embodiment of the method scripts the image creation process in HTML to create a dynamic Web site. There is no need to create over 100 images. The claimed system generates images on demand. The Web site only needs to create original assets. The scripting process involves writing the proprietary scripts. In the current example herein, scripting buttons is very simple. Once one button is created, simply copy and paste the HTML to create another button or many buttons. Only the name of the image to be overlaid on the button must be changed. The Webmaster then creates a simple program that reads what object a user has clicked on and generates a proprietary tag. The tag is then sent to the claimed system to generate a center image.

The claimed method allows the creation of all 100 combinations automatically. When the Web site receives an updated image, only the original image needs to be updated. Any change to the original image automatically propagates throughout the system. The Web site is easier to manage. Testing of the Web site is easier because there is no need to test all 100 combinations. A small subset of combinations will guarantee adequate coverage.

Processing of an Image Tag Example (FIG. **13-16**).

FIG. **13** shows two original images **1300** and **1310** to be processed according to a preferred embodiment of the invention.

FIG. **14** shows a portion on an HTML document with a proprietary tag **1400**, <freerideimage></freerideimage> according to a preferred embodiment of the invention. The use of the term “freeride” refers to an internal code name for the invention.

FIG. **15** shows an HTML document **1500** as viewed in a browser and an HTML document source **1510**, according to a preferred embodiment of the invention. The use of the term “freeride” refers to an internal code name for the invention.

FIG. **16** shows a generated GIF image **1600** according to a preferred embodiment of the invention.

Automatic Media Delivery System Operating in Parallel with Existing Web Site Infrastructure.

It should be noted that the words, media, graphics, and images are used herein interchangeably.

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, the generated media is cached so that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

A preferred embodiment of the invention is described with reference to FIG. **17**. FIG. **17** is a schematic diagram of an image system within a typical Web infrastructure according to the invention. The image system **100** is placed in parallel to an existing Web server **110**. The image system **100** may be on-site or off-site to the Web infrastructure. Multiple client browsers **120***a*-**120***d* communicate with both the Web server **110** and the image server **100** via the Internet **130**.

The delivery of an HTML document and media according to a preferred embodiment of the invention is described with reference to FIG. **18**. Resource locators (URLs) are placed within HTML documents **301** accessible to the Web server **110**. These URLs direct browsers to generate requests for media to the system **100**. The system processes such URLs by interpreting the proprietary tags, executing the indicated image generation procedures on the original media **200**, and returning derivative Web-safe media to client browsers **120***a*-**120***d* via the Internet **130**. Additionally, such generated media is cached on the image server **100** and, therefore need not be regenerated for subsequent requests.

Web site administration according to another preferred embodiment of the invention is described with reference to FIG. **19**. FIG. **19** is a schematic diagram showing the components of Web site administration according to a preferred embodiment of the invention, whereby Web site administration is simplified. The preferred embodiment provides, but is not limited to the following services: asset management, automatic image manipulation, automatic image conversion, automatic image upload, automatic image customization based on browser characteristics, automatic disk management, automatic control of proxy caching, and image delivery **501**.

FIG. **20** is a simple overview showing components of the system according to a preferred embodiment of the invention. HTML pages with proprietary URL tags **301** describe how referenced media therein is to be manipulated for Web. Browsers **120** send such tags to the image system **100** as media requests. A server **2000** within the image system **100** receives the media requests, decodes the URL tags, and retrieves any media that already exists in the media caching system **2010**. Non-existent media is subsequently generated by a media creation system **2020** using original media **2050** stored in a media repository **2040** and using content generation procedures **2030**.

The Image System.

Following is a detailed description of the preferred embodiment of the invention with reference to FIG. **21** below.

The system receives a request for media through a URL containing proprietary tags for controlling image generation. The system parses this URL to determine the content generation procedure to execute, input to the content generation procedure, post-processing directives for, for example, zoom/pan/slice, browser properties, and any cache control directives. Such data is handed to a media caching subsystem that returns the requested image if found. If the image is not found, the information is handed to the media generation subsystem that executes the specified content generation procedure to produce a derivative image. The media generation subsystem returns one or more images to the media caching system for subsequent reuse.

The media caching subsystem is a mechanism for associating final or intermediate derivative media with the procedure, input, and user characteristics used to generate said media, specified through proprietary tags within the requested URL. This system may be implemented using a database, file system, or any other mechanism having capability to track such associations.

The media generation subsystem executes a primary content generation procedure to produce a derivative image whose identifier is provided to the media caching subsystem. This derivative image is composed of one or more original images acquired from the media repository. This media is then passed to the dynamic image content system, if necessary, to generate a subsequent derivative media suitably modified for the needs of zooming, panning, or slice. The resulting media is passed to the user profile system where it is again modified to account for any specific user browser characteristics specified using the proprietary URL tags. This media is then returned to the browser, along with any cache control directives encoded within the URL, and its identifier is passed to the media caching system for subsequent retrieval.

The dynamic content system operates on intermediate derivative images to generate image subsets or scalings used by Web site designers to implement zooming in on an image, panning across an image, slicing an image into parts, and the like for special Web page effects. The input to this system is cached by the media caching system such that the intermediate image need not be regenerated.

The user profile system operates on the final image about to be returned to the browser and may modify the image to account for individual needs of Web site users. The designer of a site is able to implement freely custom post-processing of images to meet the specific needs of their clients.

FIG. **21** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention. Original media **200** is created and placed into the system **100** in a media repository **2040**. A content generation procedure **2140** is created with instructions on how the media is to be transformed to create the desired Web page content. An HTML page **301** is created for the Web site comprising the system **100**, the page containing one or more URLs directing a browser **120** to request the specified content generation procedure **2140** from the system **100** using input parameters specified with proprietary tags encoded within the URL. The browser **120** requests the Web page **301** from the Web site **110**. Upon receipt of the page **301**, the browser contacts the system **100** requesting media specified in the URL. The system parses the URL **2100** to determine the content generation procedure **2140** to execute, any corresponding input parameters to be used by such procedure, any dynamic content processing **2150** to be performed by dynamic media procedures, any user profile information **2160** to be used to modify the resulting image, and any cache control HTTP headers **2190** the site instructs to accompany the resulting image.

The parser generates a unique primary lookup key **2110** for the specified resulting media. If the key corresponds to an existing generated media **2180**, such media is returned immediately to the browser **120** through a media cache **2120**, and the transaction is complete. Otherwise, a media generation occurs. In the case of media generation requiring dynamic content processing, a unique secondary lookup key corresponding to intermediate media is generated **2130**. If intermediate media **2170** corresponding to this key is found, such media is passed directly to the dynamic media content system **2150** having dynamic media procedures, wherein appropriate action is taken to generate the required derivative from the intermediate media data. A unique key is generated for the derivative **2130** and passed to the media caching system **2120**. If the media caching system finds no such intermediate image, such intermediate image is generated according to instructions specified by the content generation procedure, cached by the media cache system **2120** as a secondary cached media **2170**, and passed to the dynamic media system **2150**. Again, appropriate action is taken to generate the required derivative from the intermediate image data.

The resulting image after any dynamic media processing is complete, is checked to ensure that the image is in a valid Web image format. If not, the image is automatically converted into a valid format.

The final media is passed to a user profile system **2160** wherein browser characteristics specified through proprietary tags within the URL are inspected, and appropriate modification to the media is performed, based on such characteristics. The resulting image is handed to the media cache system **2120** for caching and returned to the browser **120**.

FIG. **22** shows a flowchart of the content generation procedure according to a preferred embodiment of the invention. A URL containing proprietary tags (**2200**) is parsed (**2210**) to determine the content generation procedure to execute, any dynamic modifications to the media, user profile characteristics, and proxy-cache control. A unique final lookup key is generated for the media (**2220**) and the media cache is checked (**2230**). If the indicated media exists, control passes to proxy-cache control (**2290**) and the media is delivered to the browser (**2295**). Otherwise, dynamic media system tags are separated from content generation control tags (**2240**) and a unique intermediate image lookup key is generated (**2250**). The cache is then checked for such intermediate media (**2261**). If such intermediate media is found, it is used directly for dynamic processing, if required. Otherwise, content is generated (**2262**) and cached (**2263**), and the result is evaluated for dynamic processing (**2270**). If dynamic processing is required, the media is operated upon by the dynamic content generator (**2271**), otherwise it is evaluated for valid content type (**2272**). If the content type is invalid, the media is automatically converted to a valid type (**2273**). The resulting image is then customized by the user profiling system (**2280**) for specified browser or client attributes. Finally, any cache-control directives specified are attached to the response (**2290**) and the media is delivered to the browser (**2295**).

FIG. **23** is a flow chart showing an authoring process according to a preferred embodiment of the invention. The process starts (**2300**) when a user adds an original graphic or other media (**2310**) to the system. The author then creates a content generation procedure (**2320**) containing instruction on how the original media should be processed to generate the desired Web page content. The user then creates an HTML document (**2330**) that refers to that image by using a URL pointing to a content generation procedure on the image server. The system ends (**2340**). The authoring subsystem assists the Web site designer with choosing parameters and with designing the content generation procedure such that the desired Web site graphic is obtained.

It should be appreciated that differences exist between specifying an image with a URL and requesting an image using a content creation process that interprets proprietary parameters encoded within a URL. That is, URLs allow Web site designers to load specific graphic images into a Web page. In contrast and according to the invention, URLs containing proprietary content creation tags initiate a process whereby graphic images for a site are automatically produced.

Table D below is a list of example proprietary URL tags used for content generation within the system according to the preferred embodiment of the invention. Additional tags may be added to the system as necessary.

TABLE D

Tags
f=function
Names the content creation procedure used to generate all or part of the desired graphic.
args=arguments
Supplies page dependent parameters used to control the content creation procedure from within the Web page.
cr=crop rectangle
Specifies that portion of the image generated by the content generation procedure to be returned to the browser.
st=slice table
Specifies a rectangular grid to be placed over the image produced by the content generation procedure, each portion of which can be returned to the browser.

TABLE D-continued

Tags
sp= slice position Specifies that portion of the slice table grid placed over the image generated by the content creation procedure to be returned to the browser.
is=image size parameter Specifies scale factors to be applied to any portion of an image generated by any combination of a content generation procedure, arguments, crop rectangles, slice tables, and slice positions.
p=user profile string Specifies a user profile identifier used to modify the final image prior to returning the image to the browser, thus allowing clients to modify the image returned to the browser to account for individual browsing conditions.
c=cache control Specifies a proxy-cache control string to accompany the returned image within an HTTP header.

Table E below is a list of example supported content creation commands according to a preferred embodiment of the invention. Additional commands may be added as necessary.

TABLE E

Content Creation Commands
Adjust HSB Allows the HSB of an image to be altered.
Adjust RGB Allows the contrast, brightness, and color balance of an image to be altered.
Colorize Alters the hue of the pixels in the image to that of the specifiedcolor.
Brush Composite Composites the specified brush image onto the current target image.
Convert Converts the rasters to the specified type/bit-depth.
Crop Crops the media to the specified size.
Dropshadow Adds a drop shadow to the image, based on the alpha-channel.
Equalize Performs an equalization on the relevant components of the media.
FixAlpha Adjusts the RGB components of an image relative to its alpha-channel.
Flip Flips the media vertically or horizontally.
Glow Produces a glow or halo around the image.

TABLE E-continued

Content Creation Commands
Load Loads in a media from the specified file.
Normalize Similar to equalize, but for audio.
Reduce Reduces the image to a specified palette.
Rotate Rotates the media clockwise by the specified angle in degrees.
Save Saves the media to the specified file.
Scale Scales the media to the specified size.
SetColor Allows the background color, foreground color, and transparency state of the media to be set.
Text Drawing Composites the specified text onto the image.
Text Making This command, instead of compositing text onto the target, creates a new image that just encloses the text.
Zoom Zooms in on a specified portion of the media, and fits it to the specified size. Effectively this constitutes a crop followed by a scale.

Table F below lists comprises some, and is not limited to all major features of a preferred embodiment of the invention. Additional features may be added as necessary.

TABLE F

System Features
Reads and writes various file formats;
Supports many image processing operations;
Dynamically creates Web images from original assets;
Dynamically creates thumbnail images;
Dynamically and efficiently creates images that can be panned, zoomed, or sliced from original assets without Browser plugins;
Automatically propagates changes in original assets throughout the Web site;
Uses an intelligent caching mechanism for both final and intermediate graphics, comprising:
Clean up cache on demand;
Eliminates orphaned Web files; and
Optimizes Web server cache by providing most recent images;
Renders True-Type fonts on server instead of browser;
Uses intelligent scaling of line drawings;
Allows Web designers to manipulate images using a combination of content generation procedures and proprietary URL tags;
Preserves original image assets;
Optimizes Web server traffic by adjusting the bandwidth of graphics;
Optimizes images for client connection speed;
Allows clients to specify the quality of images on a Web site;
Allow site-specific customized image optimizations for a variety of purposes; and
Allows Web designers to dynamically create images by manipulating proprietary URL tags in applications.

Accordingly, although the invention has been described in detail with reference to a particular preferred embodiment, persons possessing ordinary skill in the art to which this invention pertains will appreciate that various modifications and enhancements may be made without departing from the spirit and scope of the claims that follow.

The invention claimed is:

1. A method in a host computer for developing transformation processing operations to optimize media content playback across multiple playback devices connected with the host computer in a network, the method comprising:

determining capabilities of the playback devices for processing a media stream;

receiving requests from the multiple devices for concurrent playback of media content at a first quality level, wherein the quality level is measured in terms of each of the following:

a selected compression format of the media content;

a selected bit rate of the media content; and

an image resolution of the media content;

determining a set of independent transformations of the media content that fulfill the requests at the first quality level;

if transformations are required, determining whether processing resources available on the host computer are sufficient to perform the independent transformations;

if the processing resources available on the host computer are sufficient to perform the independent transformations, the method further comprises performing the independent transformations;

monitoring available bandwidth of the network; determining whether a requested set of media streams resulting from the independent transformations is transmissible within the available bandwidth of the network; and if the requested set of media streams is not transmissible within the available bandwidth of the network, determining the set of dependent transformations such that a modified set of media streams resulting from the set of dependent transformations is transmissible within the available bandwidth of the network;

if the processing resources are insufficient to perform the independent transformations, determining a set of dependent transformations that fulfill the requests at a second quality level within limits of the processing resources of the host computer;

determining whether the capability of each of the playback devices is sufficient to process a requested media stream resulting from the independent transformations; and

if not, determining whether the capability of each of the playback devices is sufficient to process a requested media stream resulting from the dependent transformations.

2. The method of claim **1**, further comprising performing the dependent transformations.

3. The method of claim **1** further comprising monitoring available bandwidth of the network; determining whether a modified set of media streams resulting from the dependent transformations is transmissible within the available bandwidth of the network; and if the modified set of media streams is not transmissible within the available bandwidth of the network, determining a revised set of dependent transformations such that a revised set of media streams resulting from the revised set of dependent transformations is transmissible within the available bandwidth of the network.

4. The method of claim **1**, wherein the second quality level is lesser than the first quality level.

5. A method in a host computer for developing transformation processing operations to optimize media content playback across multiple playback devices connected with the host computer in a network, the method comprising:

determining capabilities of the playback devices for processing a media stream;

receiving requests from the multiple devices for concurrent playback of media content at a first quality level, wherein the quality level is measured in terms of each of the following:

a selected compression format of the media content;

a selected bit rate of the media content; and

an image resolution of the media content;

determining a set of independent transformations of the media content that fulfill the requests at the first quality level;

determining whether processing resources available on the host computer are sufficient to perform the independent transformations; and

if the processing resources available on the host computer are sufficient to perform the independent transformations, performing the independent transformations to create a requested set of media streams;

monitoring available bandwidth of the network; determining whether the requested set of media streams resulting from the independent transformations is transmissible within the available bandwidth of the network; and if the requested set of media streams is not transmissible within the available bandwidth of the network, determining the set of dependent transformations such that the modified set of media streams resulting from the set of dependent transformations is transmissible within the available bandwidth of the network;

if the processing resources are insufficient to perform the independent transformations, determining a set of dependent transformations that fulfill the requests at a second quality level within limits of the processing resources of the host computer;

performing the dependent transformations to create a modified set of media streams; and transmitting the requested set of media streams or the modified set of media streams across the network;

determining whether the capability of each of the playback devices is sufficient to process a requested media stream resulting from the independent transformations; and

if not, determining whether the capability of each of the playback devices is sufficient to process a requested media stream resulting from the dependent transformations.

6. The method of claim **5** further comprising monitoring available bandwidth of the network; determining whether the modified set of media streams resulting from the dependent transformations is transmissible within the available bandwidth of the network; and if the modified set of media streams is not transmissible within the available bandwidth of the network, determining a revised set of dependent transformations such that a revised set of media streams resulting from the revised set of dependent transformations is transmissible within the available bandwidth of the network.

7. The method of claim **5** further comprising transmitting across the network a modified set of media streams resulting from the dependent transformations across the network.

8. The method of claim **5** further comprising storing a modified set of media streams resulting from the dependent transformations as media files on a storage device.

9. The method of claim **5**, wherein the second quality level is lesser than the first quality level.

10. The method of claim **5**, wherein the quality level is measured in terms of one or more of the following: a selected format of the media content, a selected bit rate of the media content, and an image resolution of the media content.

* * * * *

Exhibit D

US008495242B2

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

(10) **Patent No.:** **US 8,495,242 B2**
(45) **Date of Patent:** **Jul. 23, 2013**

(54) **AUTOMATED MEDIA DELIVERY SYSTEM**

(75) Inventors: **Sean Barger**, Mill Valley, CA (US); **Steve Johnson**, Mill Valley, CA (US); **Matt Butler**, Beaverton, OR (US); **Jerry Destremps**, Sausalito, CA (US); **David Pochron**, Cambridge, WI (US); **Trent Brown**, San Anselmo, CA (US)

(73) Assignee: **Automated Media Processing Solutions, Inc.**, Sausalito, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 76 days.

(21) Appl. No.: **12/713,637**

(22) Filed: **Feb. 26, 2010**

(65) **Prior Publication Data**

US 2010/0153495 A1 Jun. 17, 2010

Related U.S. Application Data

(60) Division of application No. 12/173,747, filed on Jul. 15, 2008, which is a division of application No. 11/269,916, filed on Nov. 7, 2005, now abandoned, which is a continuation-in-part of application No. 09/929,904, filed on Aug. 14, 2001, now Pat. No. 6,964,009, which is a continuation of application No. 09/425,326, filed on Oct. 21, 1999, now Pat. No. 6,792,575.

(51) **Int. Cl.**
G06F 15/16 (2006.01)

(52) **U.S. Cl.**
USPC **709/236**; 709/246; 341/50; 708/204

(58) **Field of Classification Search**
USPC 709/263, 234, 231, 226, 219, 217, 709/204, 203, 200; 370/552, 468, 431, 353, 370/352, 330, 280, 241; 341/50; 708/204
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,088,052	A	2/1992	Spielman et al.
5,355,472	A	10/1994	Lewis
5,442,771	A	8/1995	Filepp et al.
5,530,852	A	6/1996	Meske, Jr. et al.
5,701,451	A	12/1997	Rogers et al.

(Continued)

FOREIGN PATENT DOCUMENTS

AU	A-53031/98	8/1998
EP	0747842	12/1996

(Continued)

OTHER PUBLICATIONS

Berinstein, Paula; "The Big Picture; Text and Graphics on UMI's ProQuest Direct: The Best (Yet) of Both Worlds", Mar. 1997; retrieved on Mar. 23, 2004 from website: http://infotoday.com/online/MarOL97/picture3.html, 11 pages.

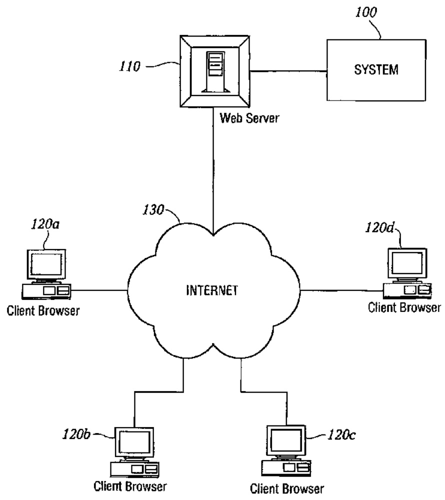
(Continued)

Primary Examiner — David Lazaro
Assistant Examiner — Charles Murphy
(74) *Attorney, Agent, or Firm* — Michael A. Glenn; Glenn Patent Group

(57) **ABSTRACT**

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed by an author within URLs embedded within Web documents.

20 Claims, 23 Drawing Sheets



U.S. PATENT DOCUMENTS

5,708,845	A		1/1998	Wistendahl et al.	
5,710,918	A		1/1998	Lagarde et al.	
5,737,619	A		4/1998	Judson	
5,745,908	A		4/1998	Anderson et al.	
5,758,110	A		5/1998	Boss et al.	
5,761,655	A		6/1998	Hoffman	
5,793,964	A		8/1998	Rogers et al.	
5,819,261	A		10/1998	Takahashi et al.	
5,822,436	A		10/1998	Rhoads	
5,845,084	A		12/1998	Cordell et al.	
5,845,279	A	*	12/1998	Garofalakis et al.	1/1
5,845,299	A		12/1998	Arora et al.	
5,860,068	A		1/1999	Cook	
5,860,073	A		1/1999	Ferrel et al.	
5,861,881	A		1/1999	Freeman et al.	
5,862,325	A		1/1999	Reed et al.	
5,864,337	A		1/1999	Marvin	
5,870,552	A		2/1999	Dozier et al.	
5,880,740	A		3/1999	Halliday et al.	
5,890,170	A		3/1999	Sidana	
5,895,476	A		4/1999	Orr et al.	
5,895,477	A		4/1999	Orr et al.	
5,903,892	A		5/1999	Hoffert et al.	
5,937,160	A		8/1999	Davis et al.	
5,943,680	A		8/1999	Ohga et al.	
5,956,737	A		9/1999	King et al.	
6,009,436	A		12/1999	Motoyama et al.	
6,311,185	B1		10/2001	Markowitz et al.	
6,456,305	B1		9/2002	Qureshi et al.	
6,463,445	B1	*	10/2002	Suzuki et al.	1/1
6,483,851	B1		11/2002	Neogi	
6,484,149	B1		11/2002	Jemmes et al.	
6,563,517	B1	*	5/2003	Bhagwat et al.	715/735
6,591,280	B2		7/2003	Orr	
6,623,529	B1		9/2003	Lakritz	
6,909,708	B1	*	6/2005	Krishnaswamy et al.	370/352
6,938,073	B1		8/2005	Mendhekar et al.	
7,284,201	B2		10/2007	Cohen-Solal	
7,313,361	B2		12/2007	Steelberg et al.	
7,406,434	B1		7/2008	Chang et al.	
7,477,688	B1		1/2009	Zhang et al.	
7,673,063	B2		3/2010	Xie et al.	
2003/0225568	A1		12/2003	Salmonsen	
2004/0025176	A1		2/2004	Franklin et al.	
2005/0091311	A1		4/2005	Lund et al.	
2005/0255852	A1		11/2005	Steelberg et al.	
2005/0278794	A1		12/2005	Leinonen et al.	
2006/0015580	A1		1/2006	Gabriel et al.	
2006/0127059	A1		6/2006	Fanning	
2007/0061198	A1		3/2007	Ramer et al.	
2007/0234213	A1		10/2007	Krikorian et al.	
2008/0155230	A1		6/2008	Robbins et al.	
2008/0186377	A1		8/2008	Eriksson et al.	
2008/0195938	A1		8/2008	Tischer et al.	
2008/0205389	A1		8/2008	Fang et al.	
2008/0207182	A1		8/2008	Maharajh et al.	
2009/0013347	A1		1/2009	Ahanger et al.	
2009/0070485	A1		3/2009	Barger et al.	
2009/0089422	A1		4/2009	Barger et al.	
2009/0240569	A1		9/2009	Ramer et al.	
2009/0254672	A1		10/2009	Zhang	
2010/0046842	A1		2/2010	Conwell	
2010/0153495	A1		6/2010	Barger et al.	
2011/0221745	A1		9/2011	Goldman et al.	
2011/0279638	A1		11/2011	Periyannan et al.	
2012/0016858	A1		1/2012	Rathod	

FOREIGN PATENT DOCUMENTS

EP	0782085	7/1997
EP	0818907	1/1998
EP	0843276	5/1998
EP	0876034	11/1998
EP	0883068	12/1998
EP	0886409	12/1998
EP	0895171	2/1999
EP	0926607	6/1999
EP	09499571	10/1999
WO	WO 97/49252	12/1997
WO	WO 98/40842	9/1998
WO	WO 98/43177	10/1998

OTHER PUBLICATIONS

Bulterman, Dick.C.A.; *Models, Media and Motion: Using the Web to Support Multimedia Documents;* Proceedings of 1997 Int'l Conf. on Multimedia Modeling; p. 17-20; Nov. 1997; Singapore.

Dobson, R.; *Animating Your Web Pages with Direct Animation;* Web Techniques; vol. 3, No. 6, p. 49-52; Jun. 1998.

McNeil, Sara; Research Interests; retrieved on Mar. 18, 2004 from website: http//www.coe.uh.edu/'smcneil/research.htm, 3 pages.

Mohler, J.L.; *Migrating Course Materials to the World Wide Web: A Case Study of the Department of Technical Graphics at Purdue University;* Computer Networks and ISDN Systems; vol. 30, Issues 20-21, pp. 1981-1990; Nov. 12, 1988.

Sakaguchi, et al.; "A browsing tool for multi-lingual documents for users without multi-lingual fonts"; 1996; ACM International Conference on Digital Libraries, pp. 63-71.

Tables of Contents service for Computers & Geosciences; Copyright 1997; Computers and GeoSciences, vol. 23, Issue 5, retrieved on Mar. 18, 2004 from website: http://library.iem.ac.ru/comp&geo/00983004/sz977014.html, 2 pages.

Zaiane, et al.; "Mining multimedia data"; Nov. 1998; ACM Conference of the Center for Advanced Studies on Collaborative research, pp. 1-18.

* cited by examiner

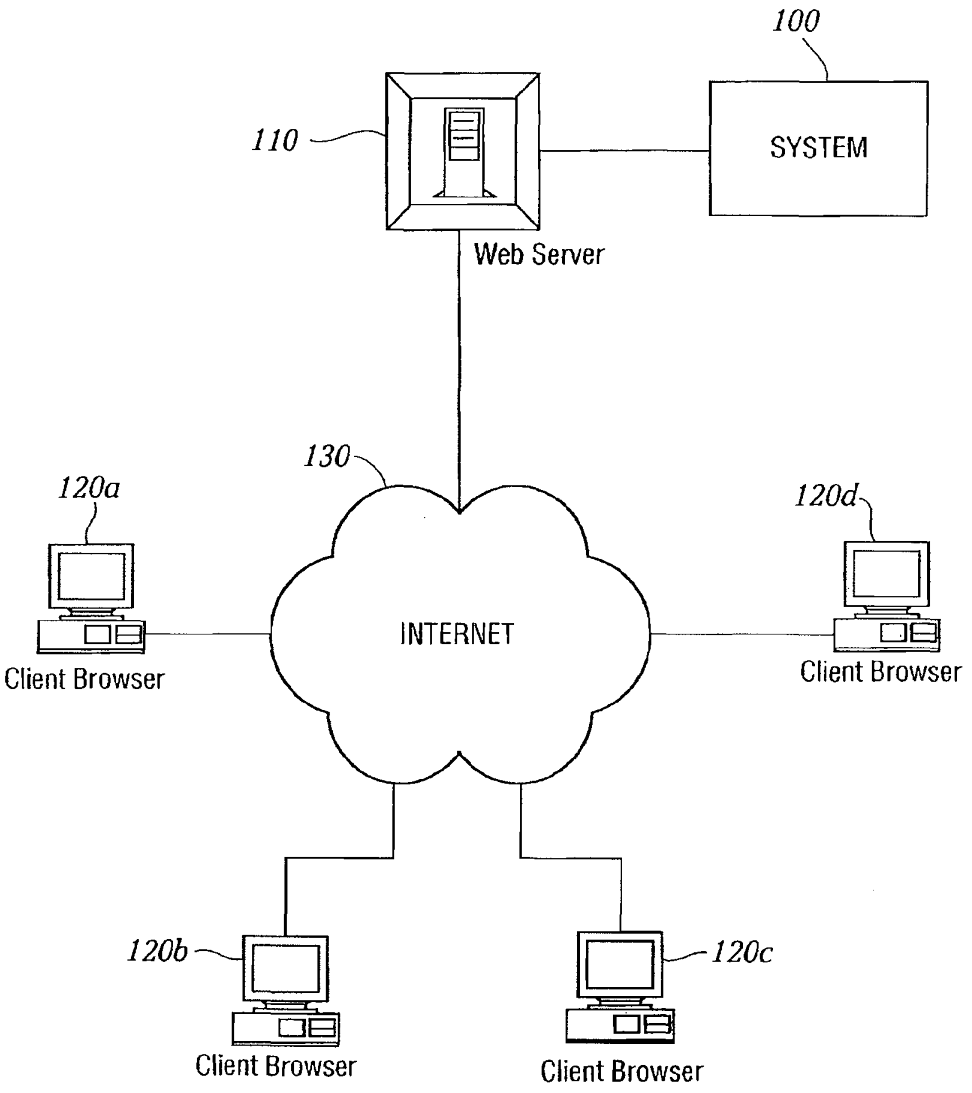

100
SYSTEM
110
Web Server
130
INTERNET
120a
Client Browser
120d
Client Browser
120b
Client Browser
120c
Client Browser

FIG. 1

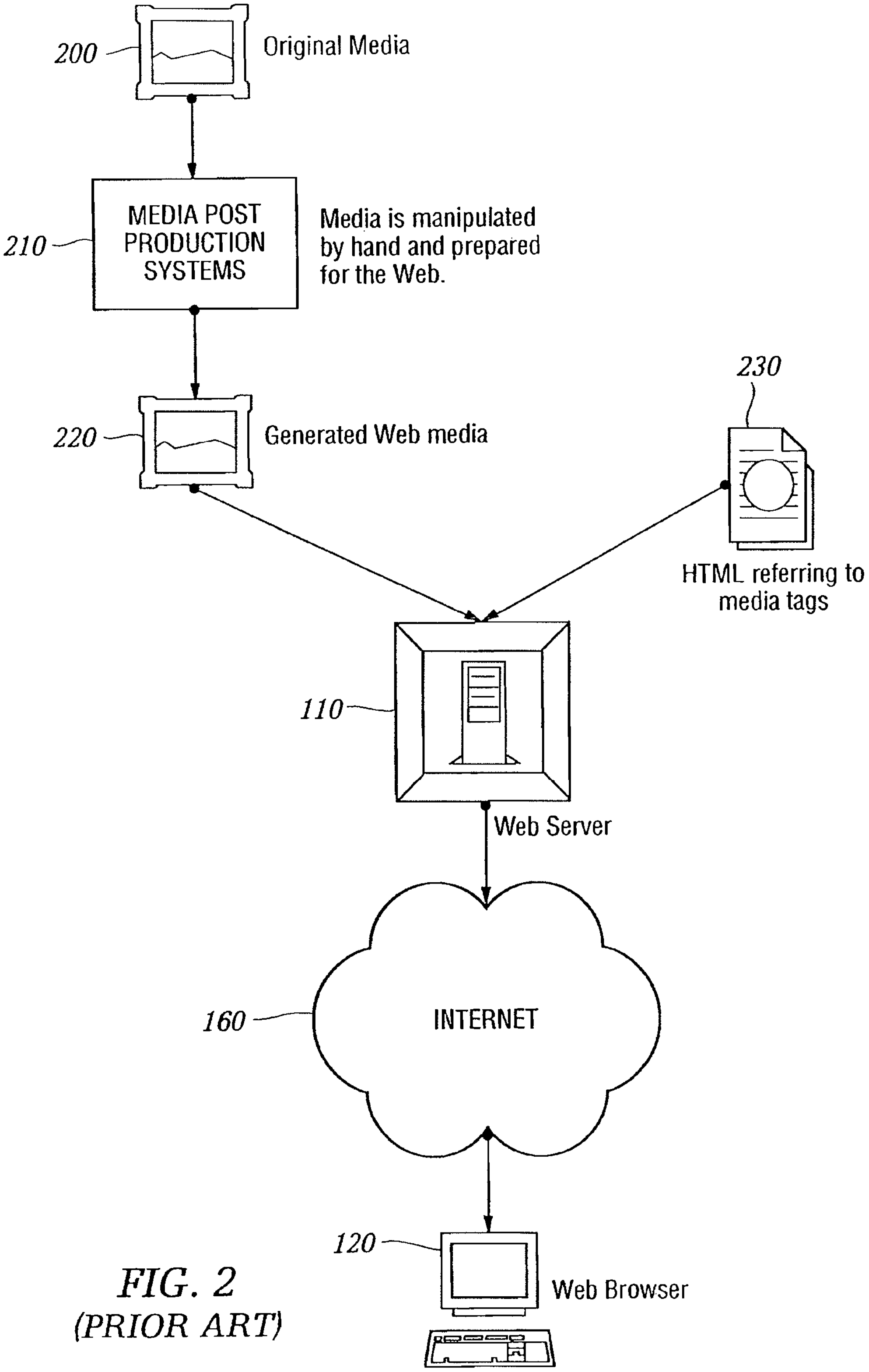

200
Original Media
210
MEDIA POST PRODUCTION SYSTEMS
Media is manipulated by hand and prepared for the Web.
220
Generated Web media
230
HTML referring to media tags
110
Web Server
160
INTERNET
120
Web Browser

FIG. 2
(PRIOR ART)

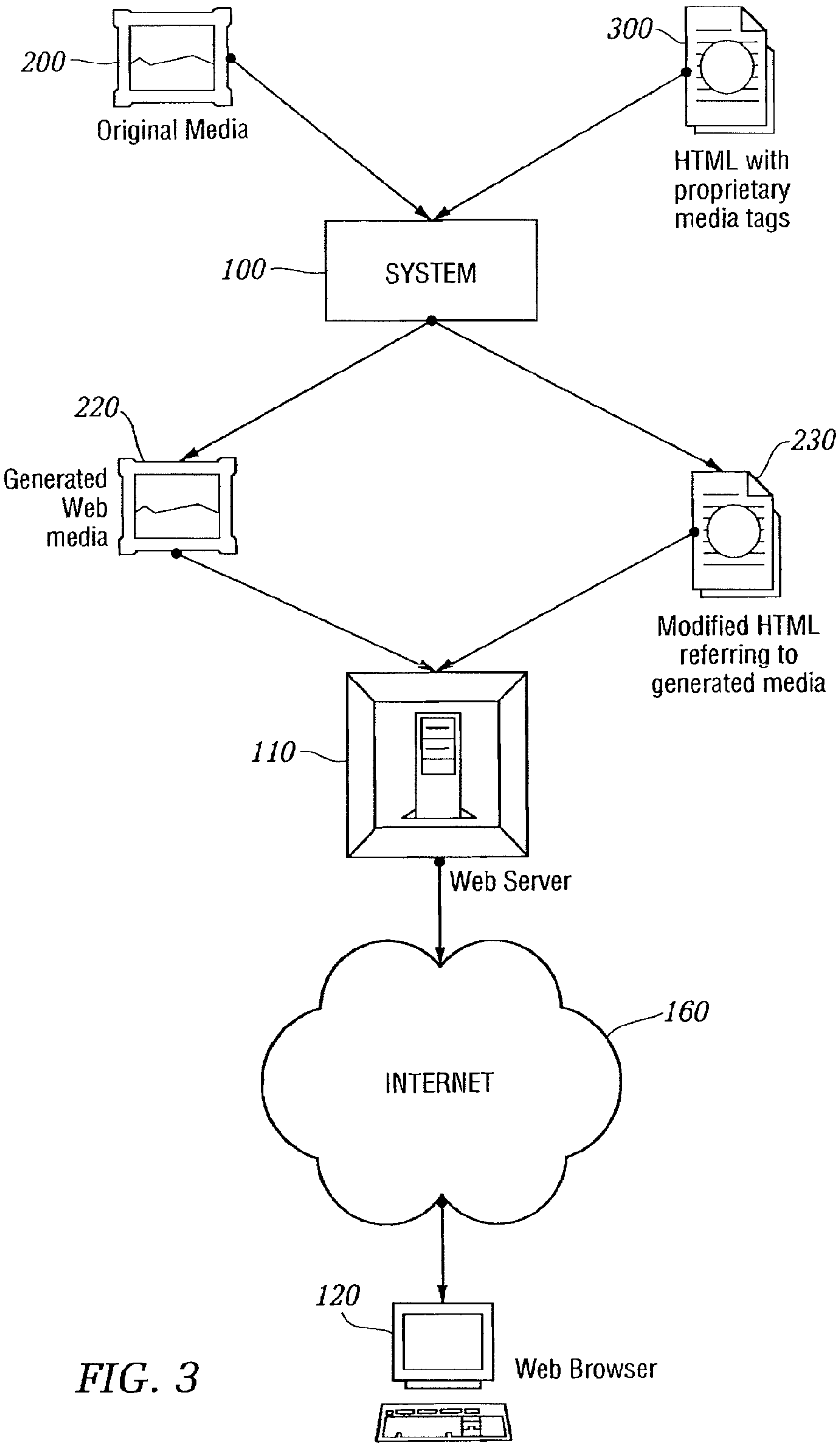

200
Original Media
300
HTML with proprietary media tags
100
SYSTEM
220
Generated Web media
230
Modified HTML referring to generated media
110
Web Server
160
INTERNET
120
Web Browser

FIG. 3

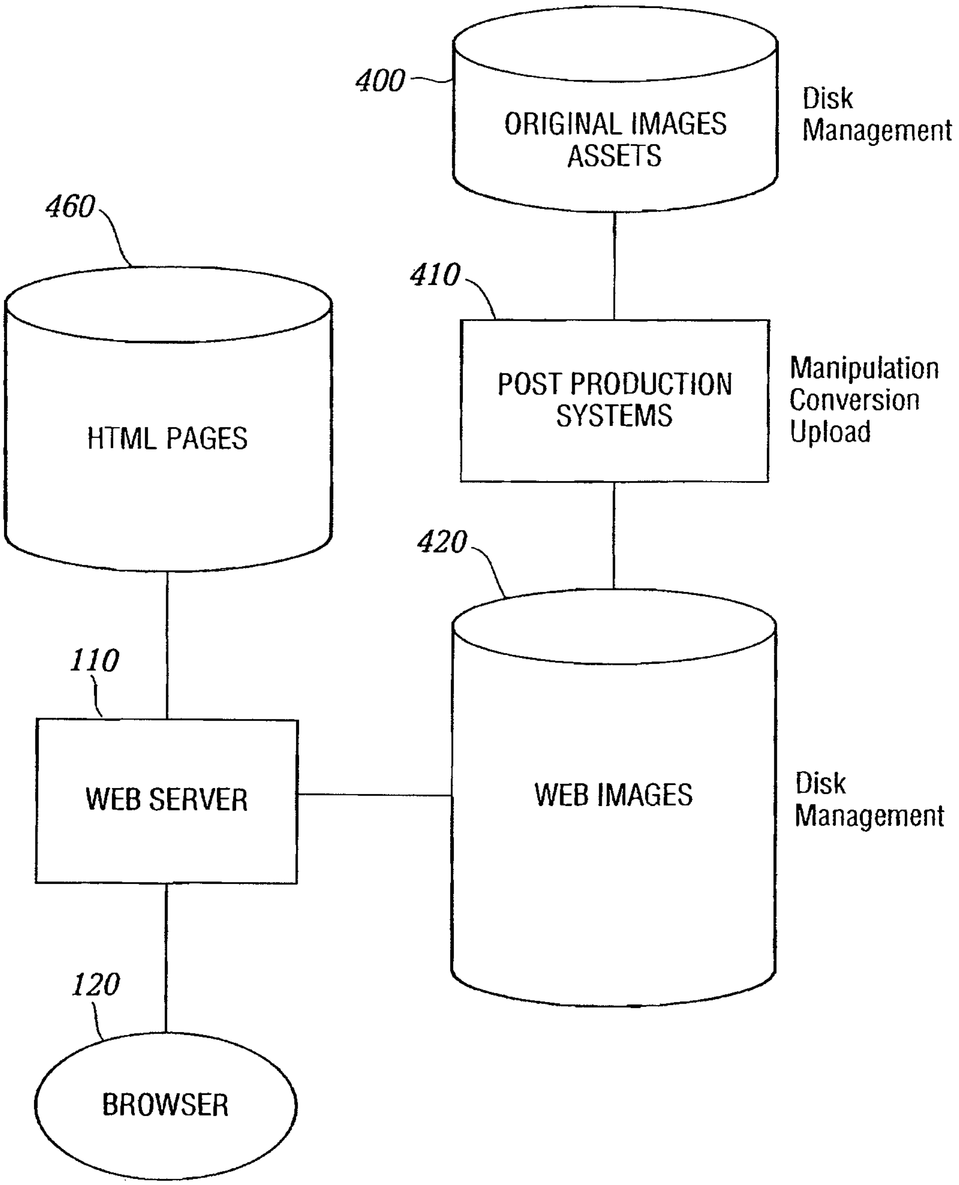


FIG. 4
(PRIOR ART)

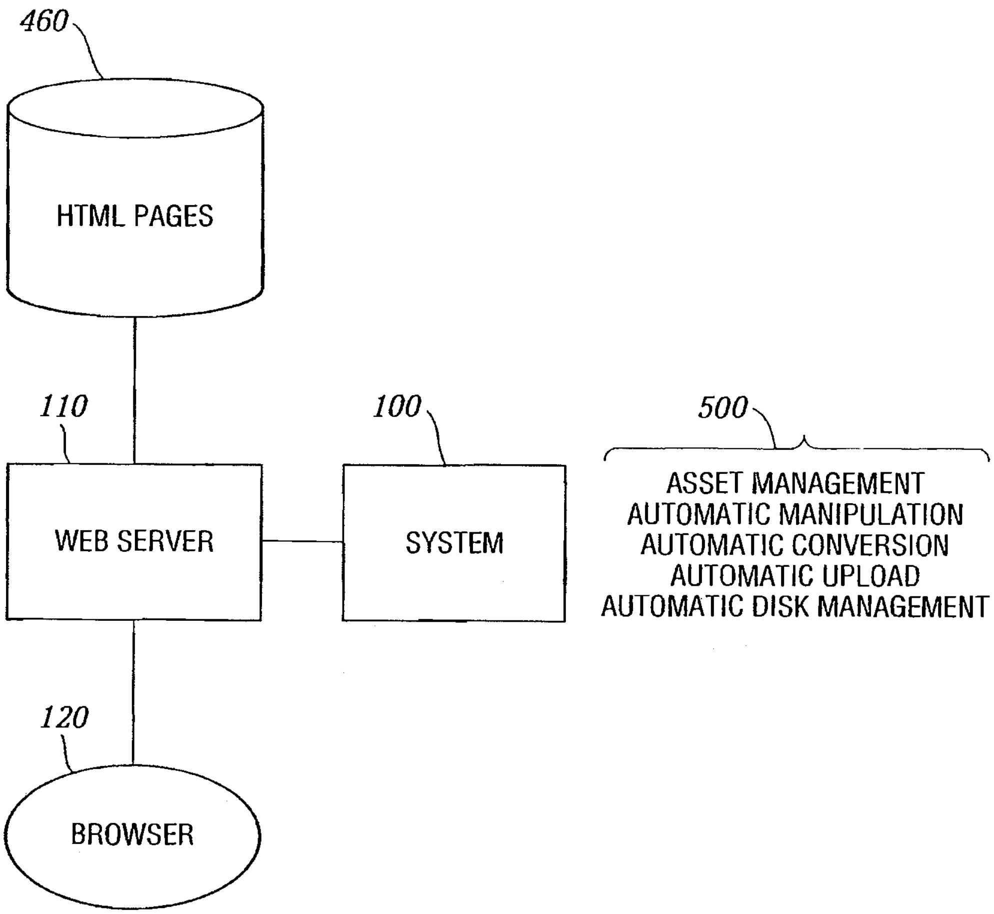

460
HTML PAGES
110
WEB SERVER
100
SYSTEM
500
ASSET MANAGEMENT
AUTOMATIC MANIPULATION
AUTOMATIC CONVERSION
AUTOMATIC UPLOAD
AUTOMATIC DISK MANAGEMENT
120
BROWSER

FIG. 5

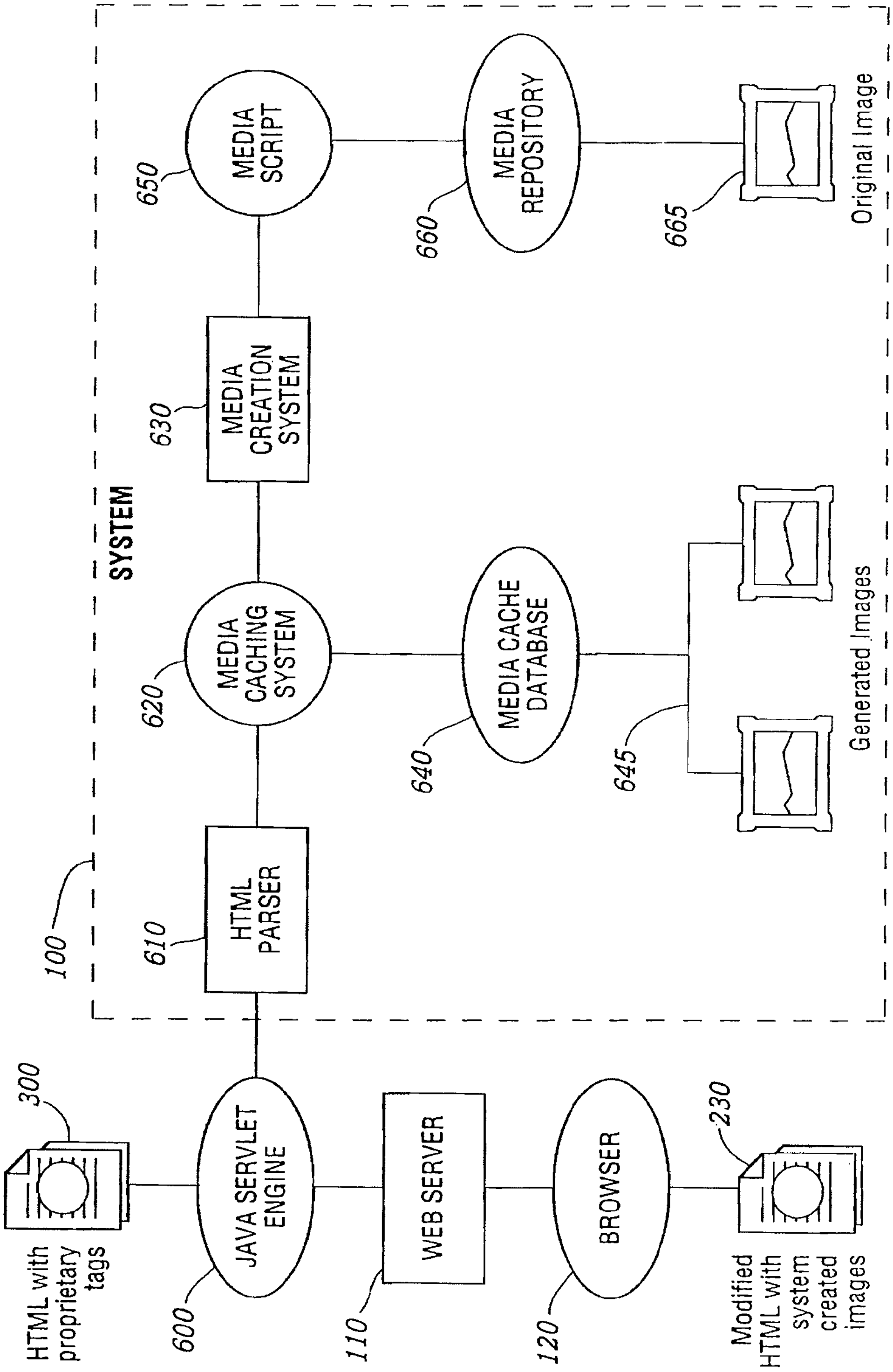

HTML with proprietary tags
300
JAVA SERVLET ENGINE
600
WEB SERVER
110
BROWSER
120
Modified HTML with system created images
230
100
SYSTEM
HTML PARSER
610
MEDIA CACHING SYSTEM
620
MEDIA CREATION SYSTEM
630
MEDIA SCRIPT
650
MEDIA CACHE DATABASE
640
MEDIA REPOSITORY
660
645
Generated Images
665
Original Image

FIG. 6

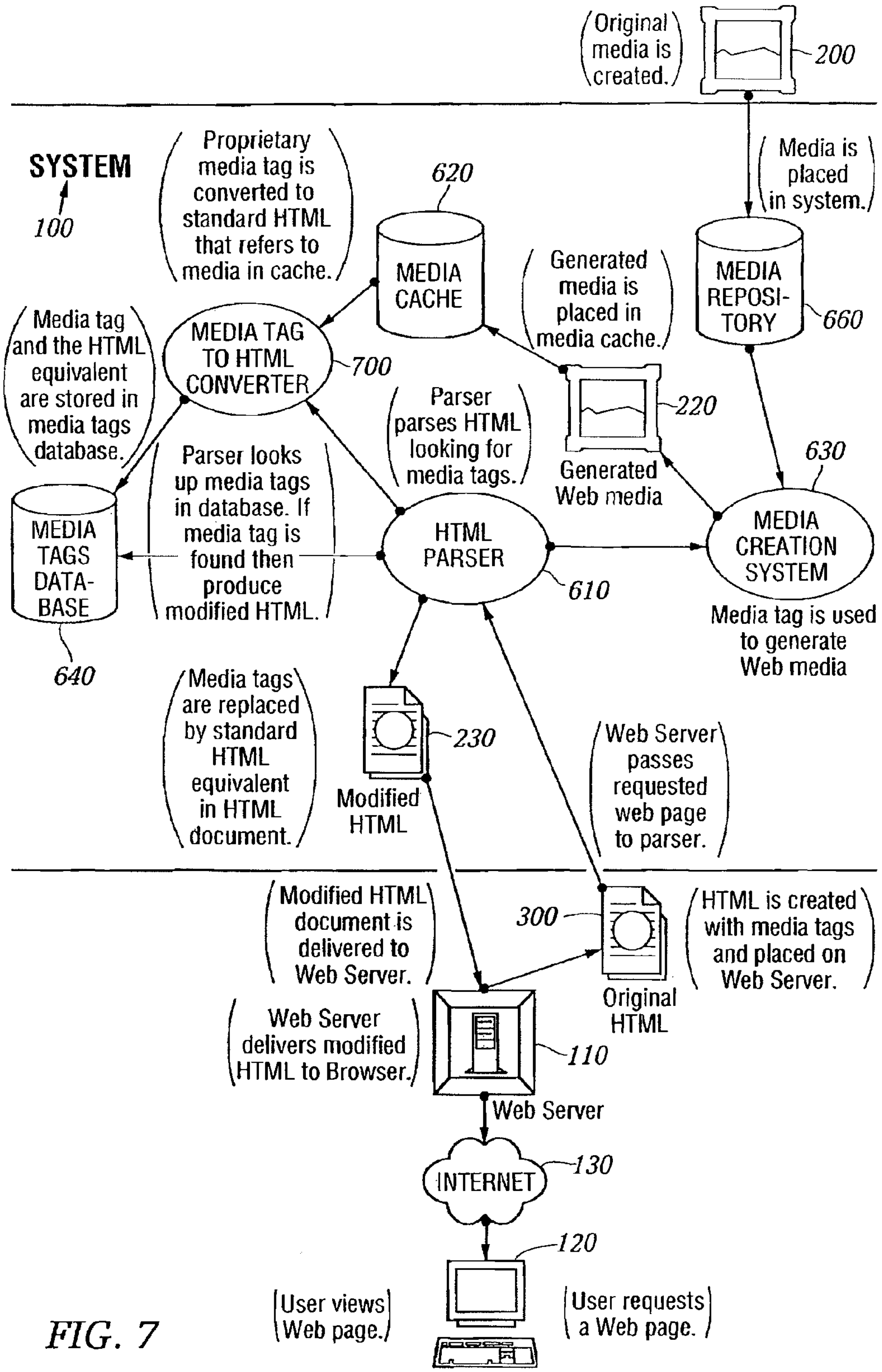

Original media is created.
200
SYSTEM
100
Proprietary media tag is converted to standard HTML that refers to media in cache.
620
MEDIA CACHE
Generated media is placed in media cache.
Media is placed in system.
MEDIA REPOSITORY
660
Media tag and the HTML equivalent are stored in media tags database.
MEDIA TAG TO HTML CONVERTER
700
Parser parses HTML looking for media tags.
220
Generated Web media
630
MEDIA TAGS DATABASE
640
Parser looks up media tags in database. If media tag is found then produce modified HTML.
HTML PARSER
610
MEDIA CREATION SYSTEM
Media tag is used to generate Web media
Media tags are replaced by standard HTML equivalent in HTML document.
230
Modified HTML
Web Server passes requested web page to parser.
Modified HTML document is delivered to Web Server.
300
HTML is created with media tags and placed on Web Server.
Original HTML
Web Server delivers modified HTML to Browser.
110
Web Server
INTERNET
130
120
User views Web page.
User requests a Web page.

FIG. 7

AUTHORING FLOWCHART

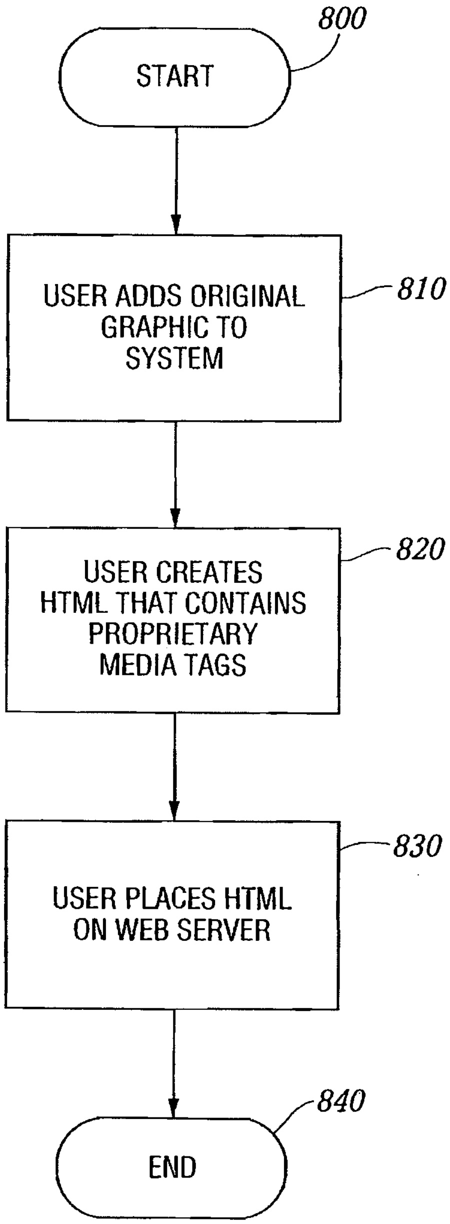

800
START
USER ADDS ORIGINAL GRAPHIC TO SYSTEM
810
USER CREATES HTML THAT CONTAINS PROPRIETARY MEDIA TAGS
820
USER PLACES HTML ON WEB SERVER
830
840
END

FIG. 8

HTML PARSING FLOWCHART

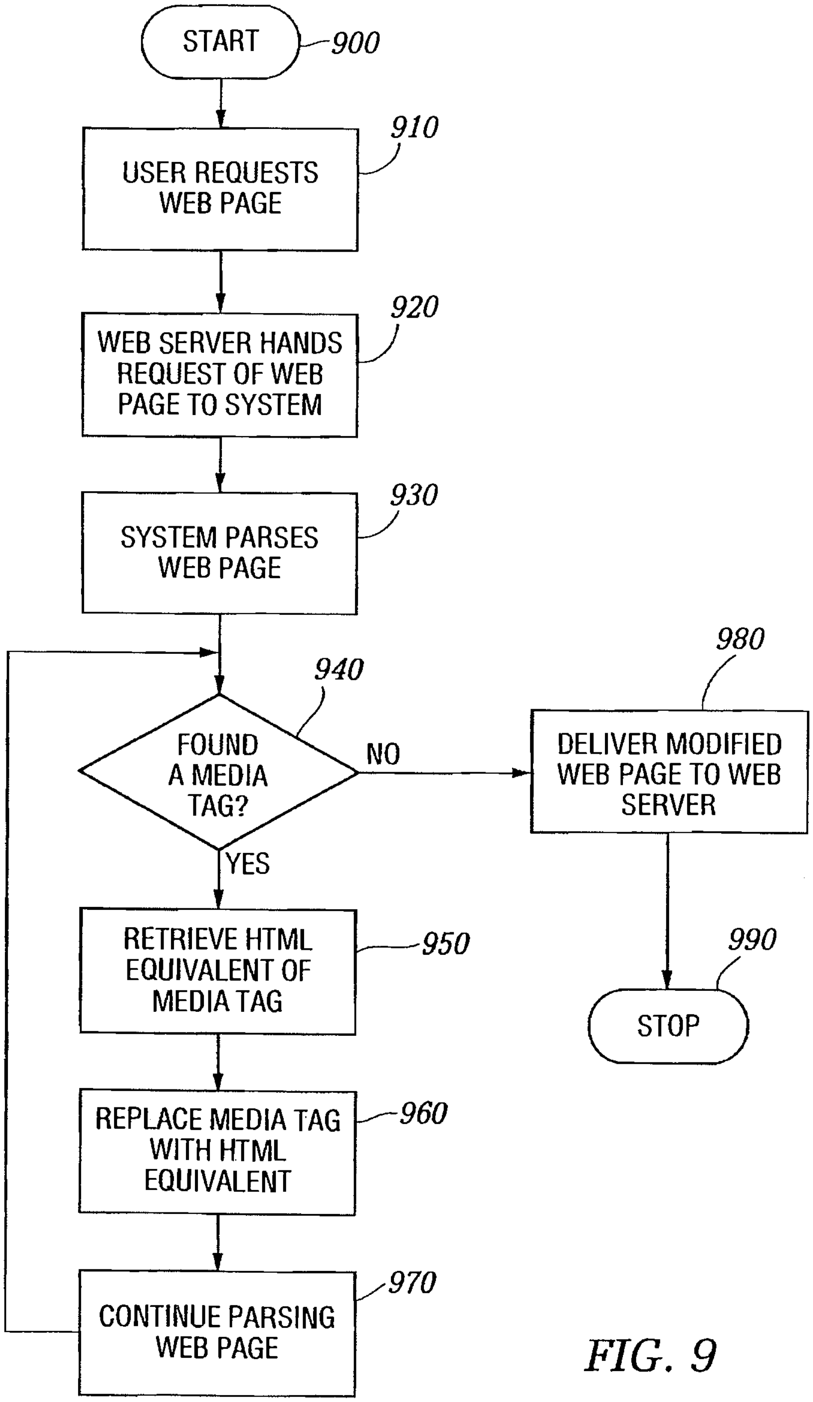

START
900
USER REQUESTS WEB PAGE
910
WEB SERVER HANDS REQUEST OF WEB PAGE TO SYSTEM
920
SYSTEM PARSES WEB PAGE
930
FOUND A MEDIA TAG?
940
NO
YES
DELIVER MODIFIED WEB PAGE TO WEB SERVER
980
STOP
990
RETRIEVE HTML EQUIVALENT OF MEDIA TAG
950
REPLACE MEDIA TAG WITH HTML EQUIVALENT
960
CONTINUE PARSING WEB PAGE
970

FIG. 9

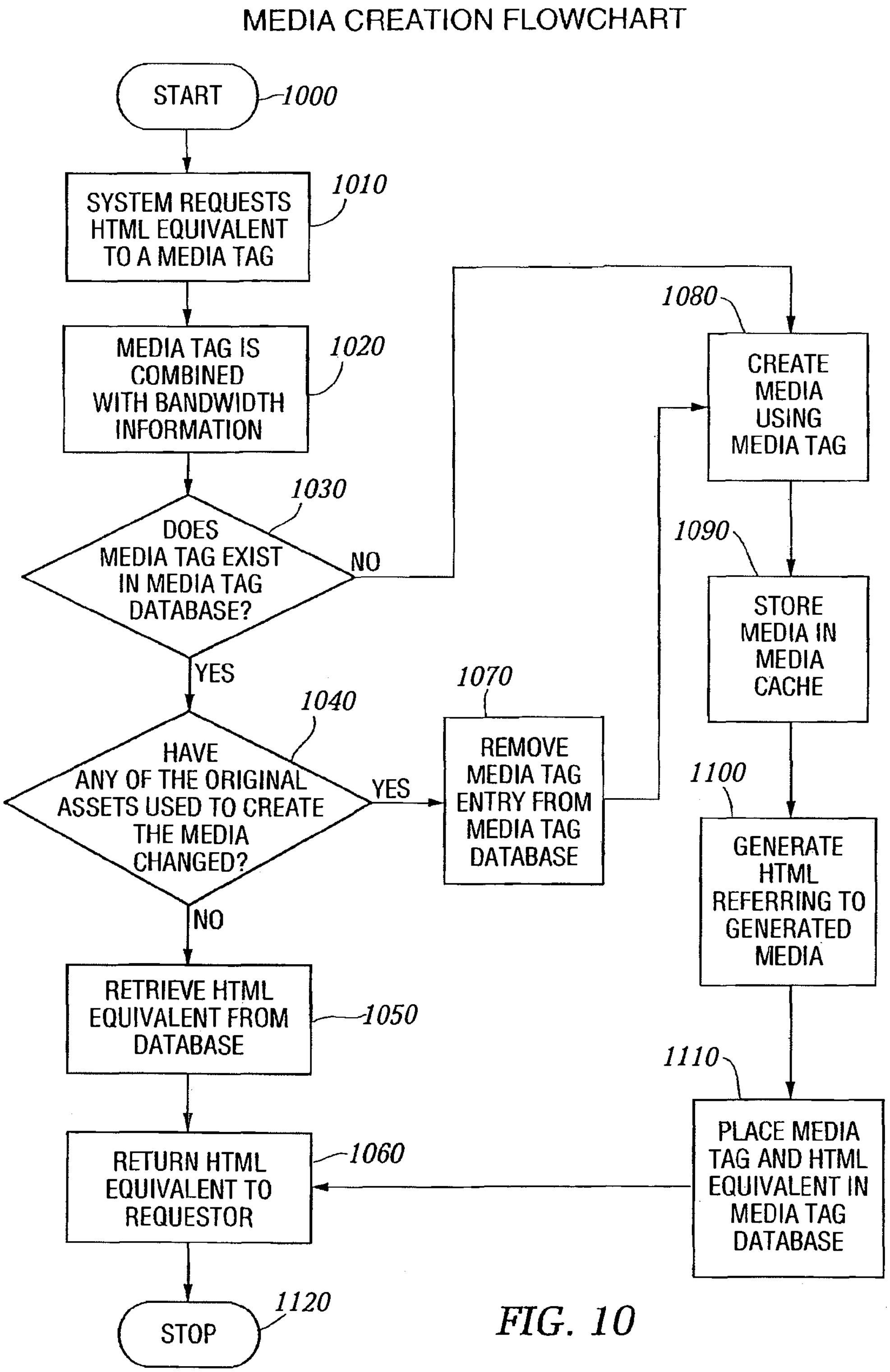
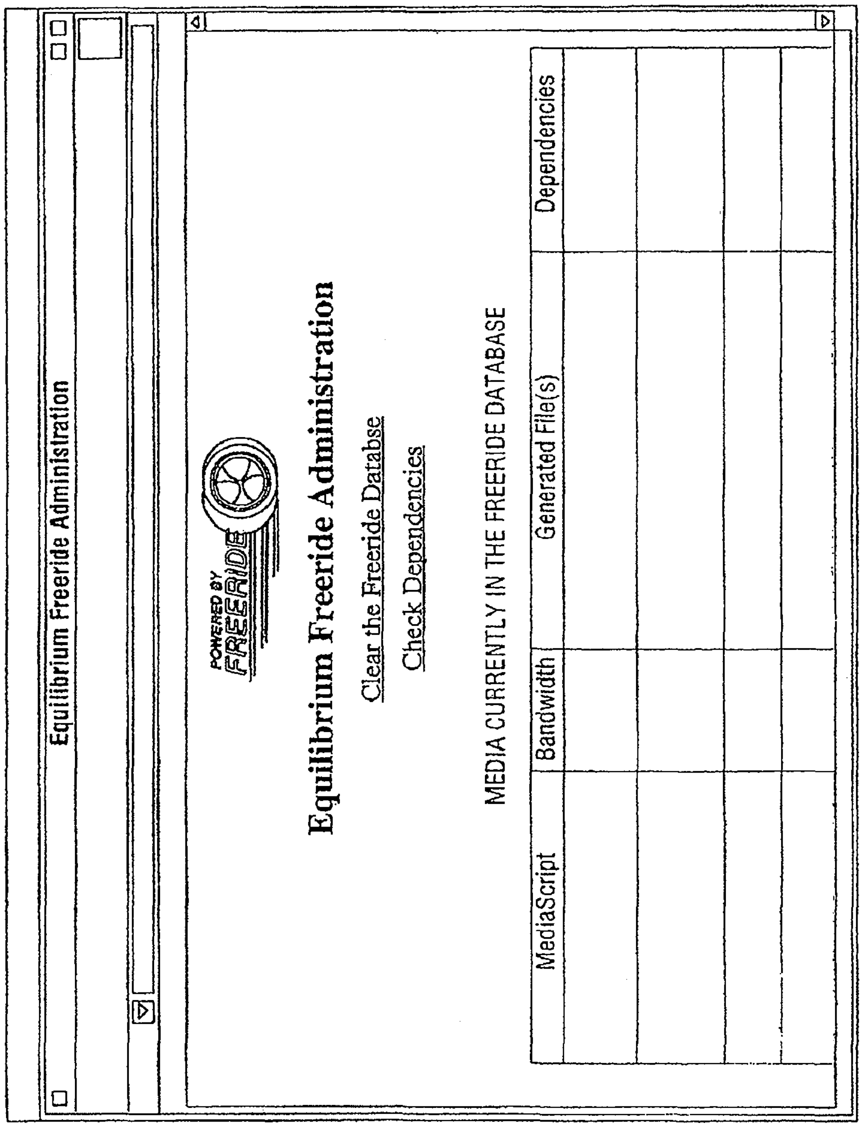

MEDIA CREATION FLOWCHART
START
1000
SYSTEM REQUESTS HTML EQUIVALENT TO A MEDIA TAG
1010
MEDIA TAG IS COMBINED WITH BANDWIDTH INFORMATION
1020
DOES MEDIA TAG EXIST IN MEDIA TAG DATABASE?
1030
NO
YES
HAVE ANY OF THE ORIGINAL ASSETS USED TO CREATE THE MEDIA CHANGED?
1040
YES
NO
REMOVE MEDIA TAG ENTRY FROM MEDIA TAG DATABASE
1070
1080
CREATE MEDIA USING MEDIA TAG
1090
STORE MEDIA IN MEDIA CACHE
1100
GENERATE HTML REFERRING TO GENERATED MEDIA
1110
PLACE MEDIA TAG AND HTML EQUIVALENT IN MEDIA TAG DATABASE
RETRIEVE HTML EQUIVALENT FROM DATABASE
1050
RETURN HTML EQUIVALENT TO REQUESTOR
1060
STOP
1120

FIG. 10



Equilibrium Freeride Administration

Equilibrium Freeride Administration

Clear the Freeride Databse

Check Dependencies

MEDIA CURRENTLY IN THE FREERIDE DATABASE

MediaScript	Bandwidth	Generated File(s)	Dependencies

FIG. 11

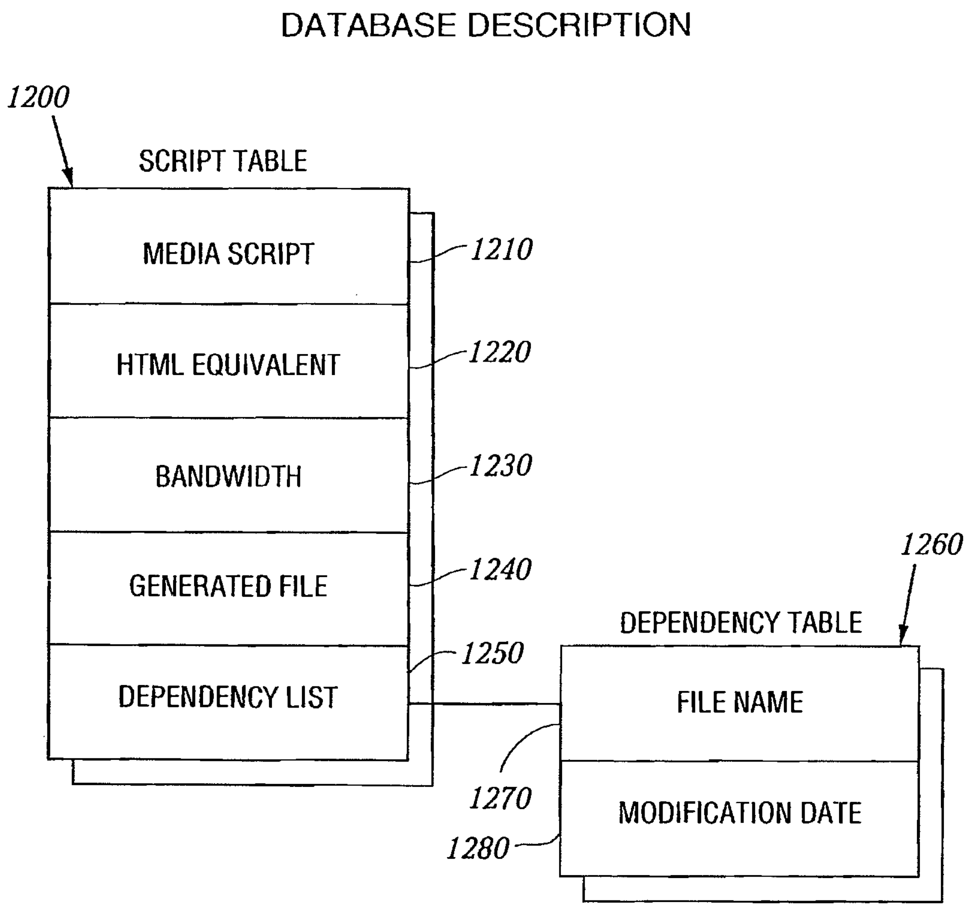

DATABASE DESCRIPTION
1200
SCRIPT TABLE
MEDIA SCRIPT
1210
HTML EQUIVALENT
1220
BANDWIDTH
1230
GENERATED FILE
1240
DEPENDENCY LIST
1250
1260
DEPENDENCY TABLE
FILE NAME
MODIFICATION DATE
1270
1280

FIG. 12

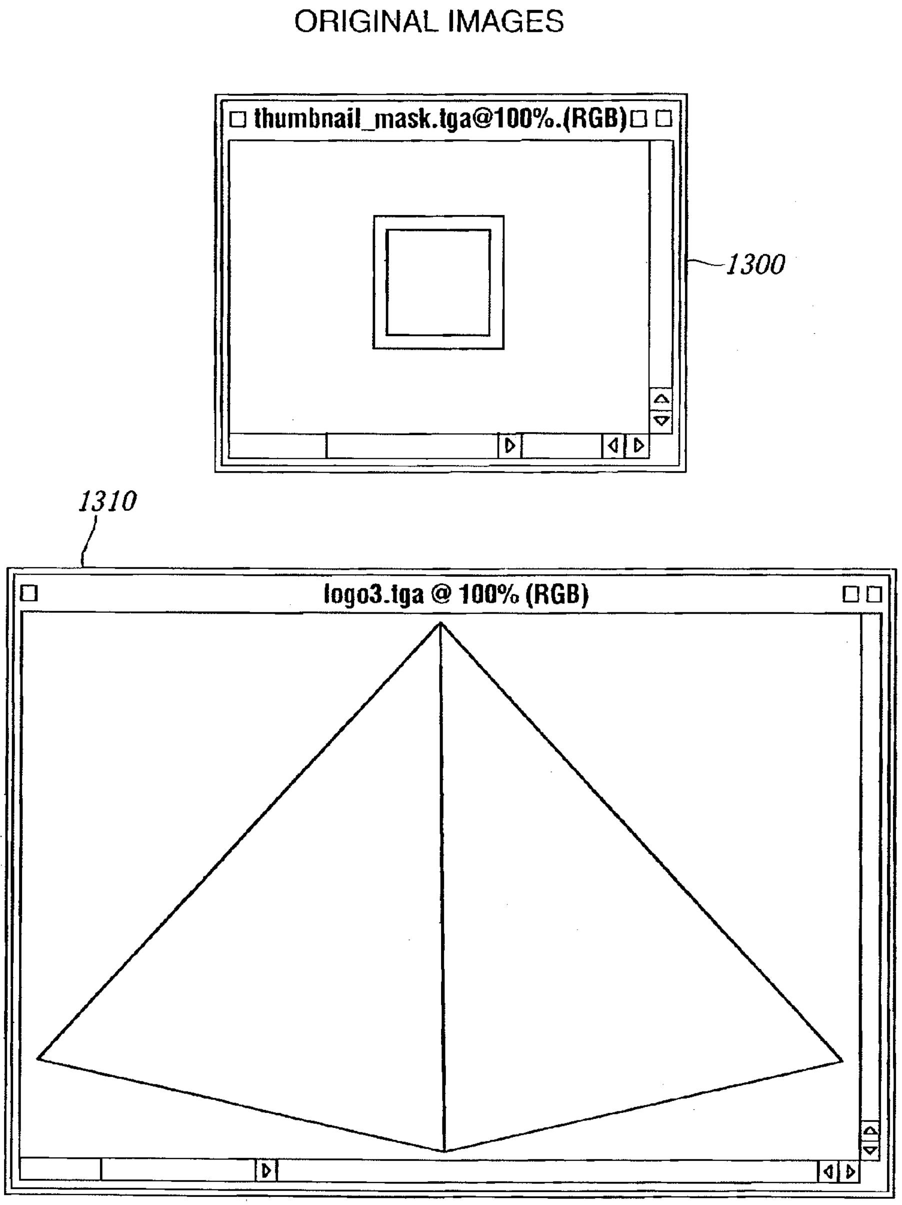

ORIGINAL IMAGES
thumbnail_mask.tga@100%.(RGB)
1300
1310
logo3.tga @ 100% (RGB)

FIG.13

HTML DOCUMENT WITH PROPRIETARY TAG

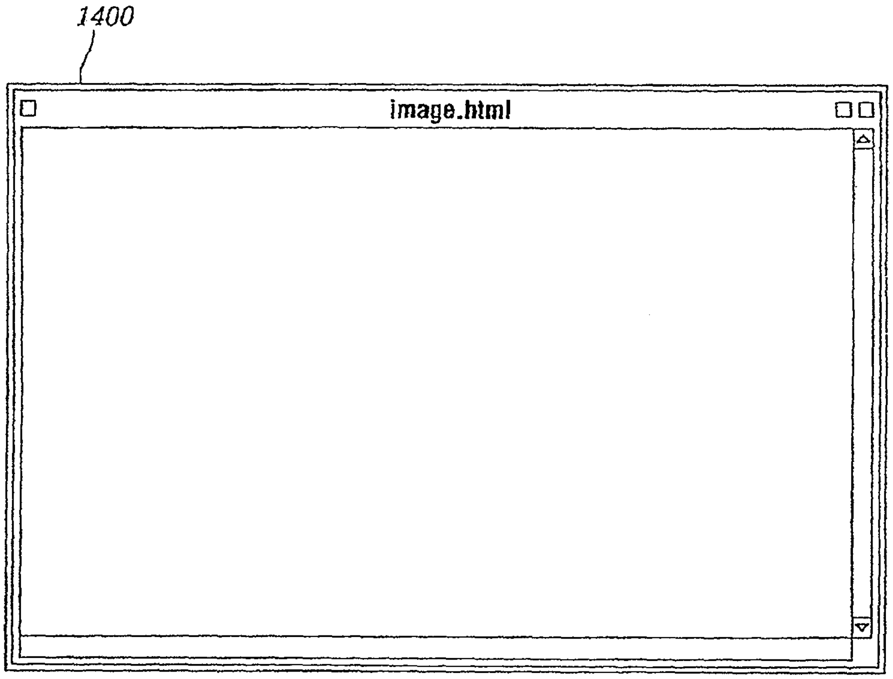


FIG.14

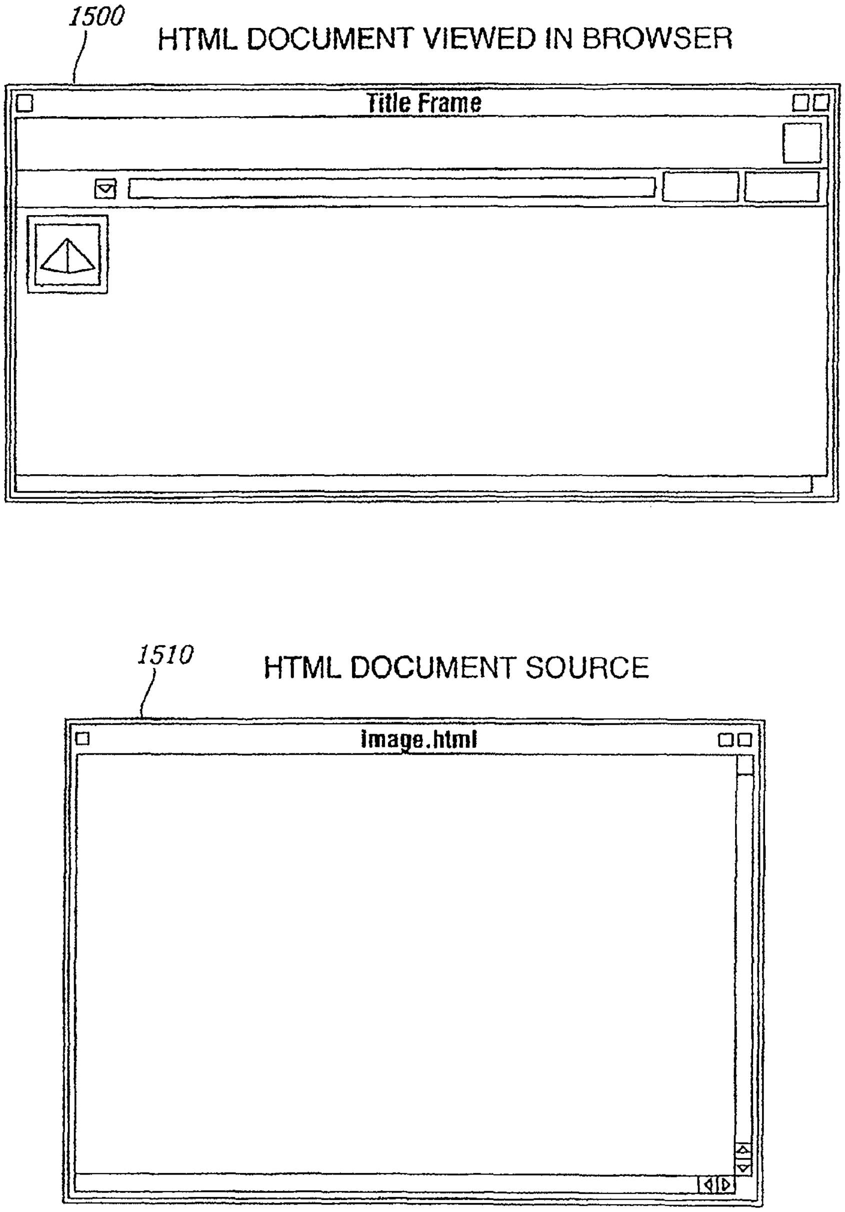

1500
HTML DOCUMENT VIEWED IN BROWSER
Title Frame
1510
HTML DOCUMENT SOURCE
image.html

FIG.15

GENERATED GIF IMAGE

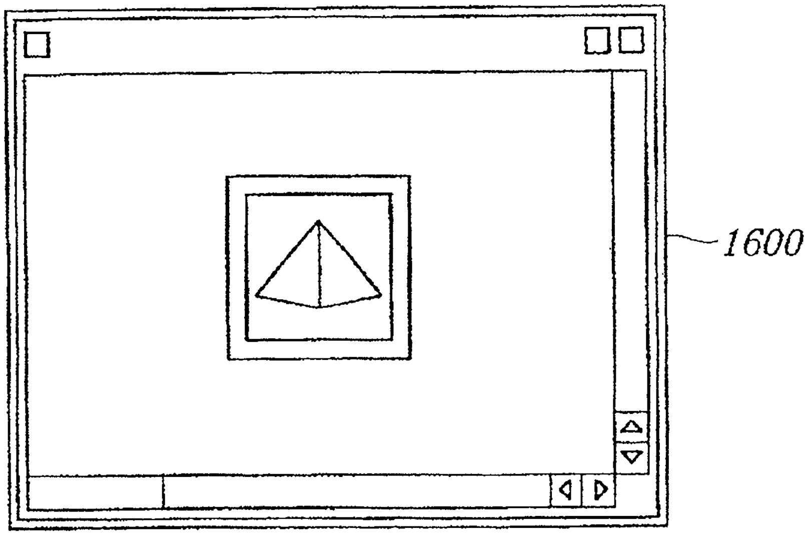


FIG.16

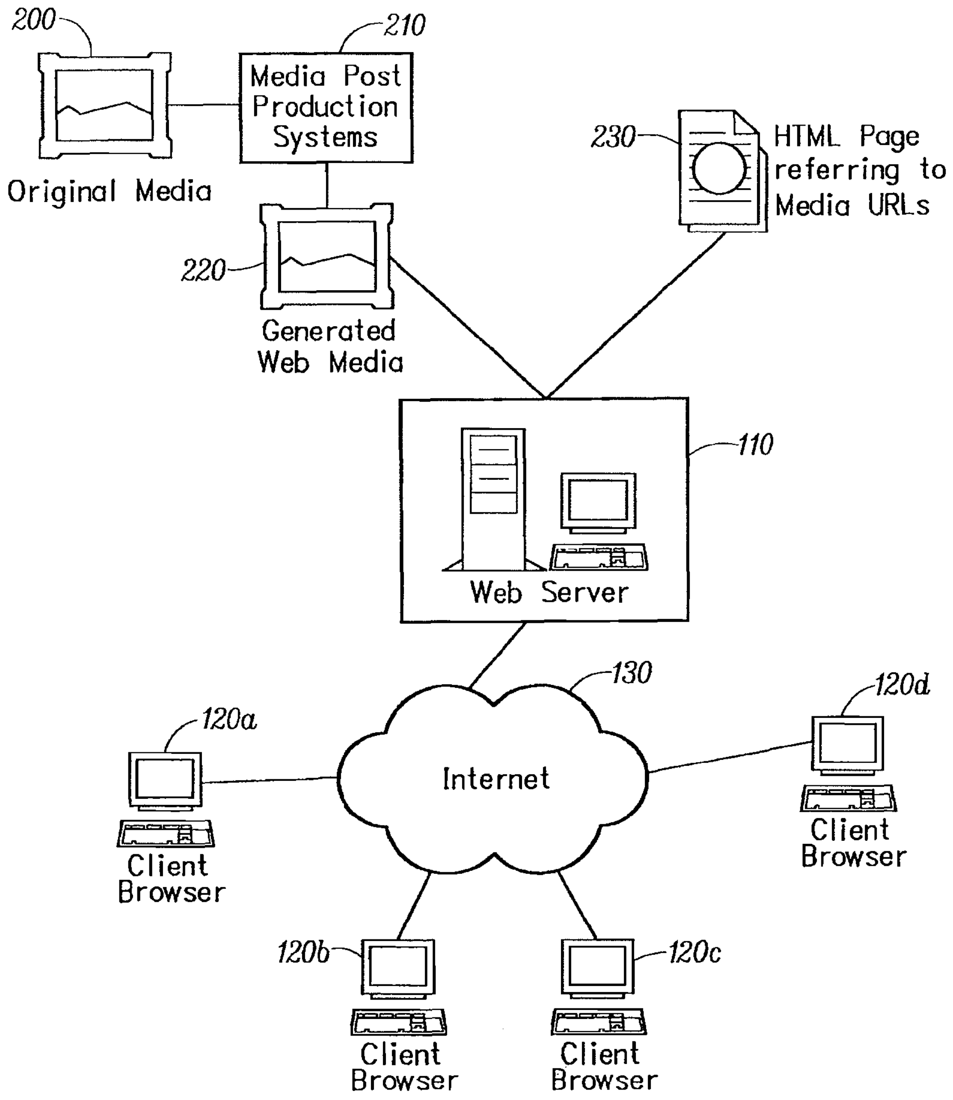

200
Original Media
210
Media Post Production Systems
220
Generated Web Media
230
HTML Page referring to Media URLs
110
Web Server
130
Internet
120a
Client Browser
120b
Client Browser
120c
Client Browser
120d
Client Browser

FIG. 17

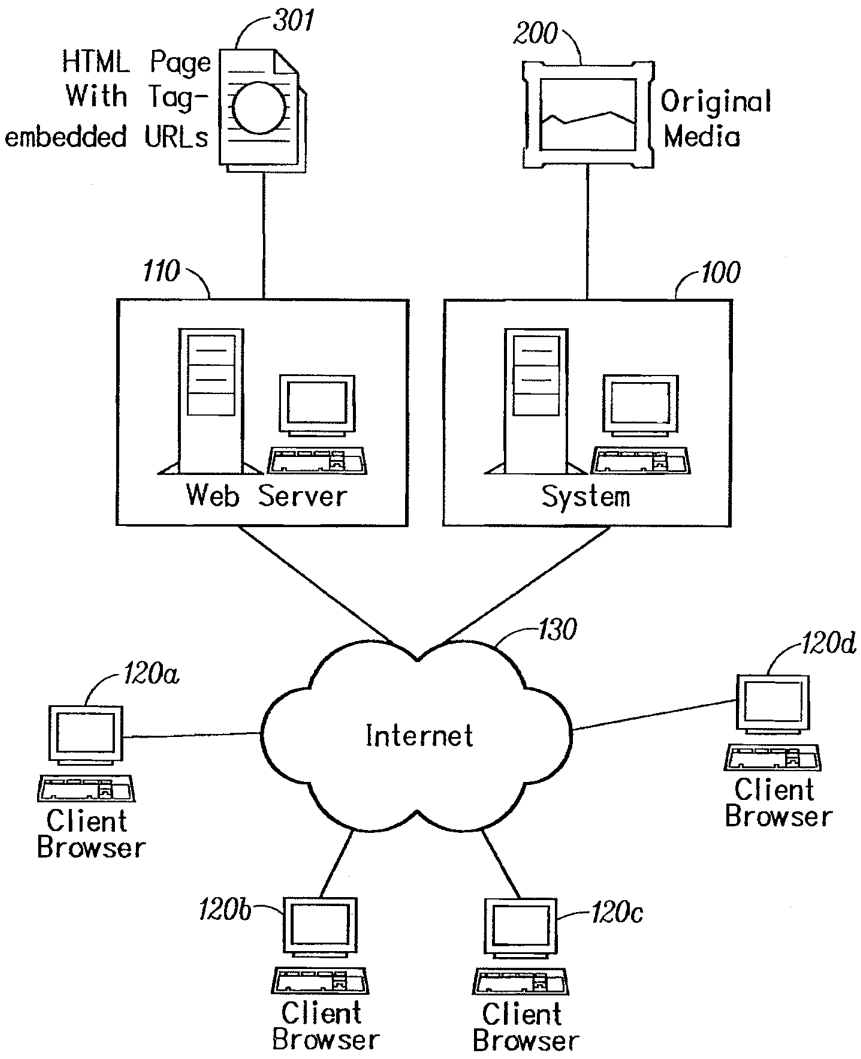

301
HTML Page With Tag-embedded URLs
200
Original Media
110
Web Server
100
System
130
Internet
120a
Client Browser
120b
Client Browser
120c
Client Browser
120d
Client Browser

FIG. 18

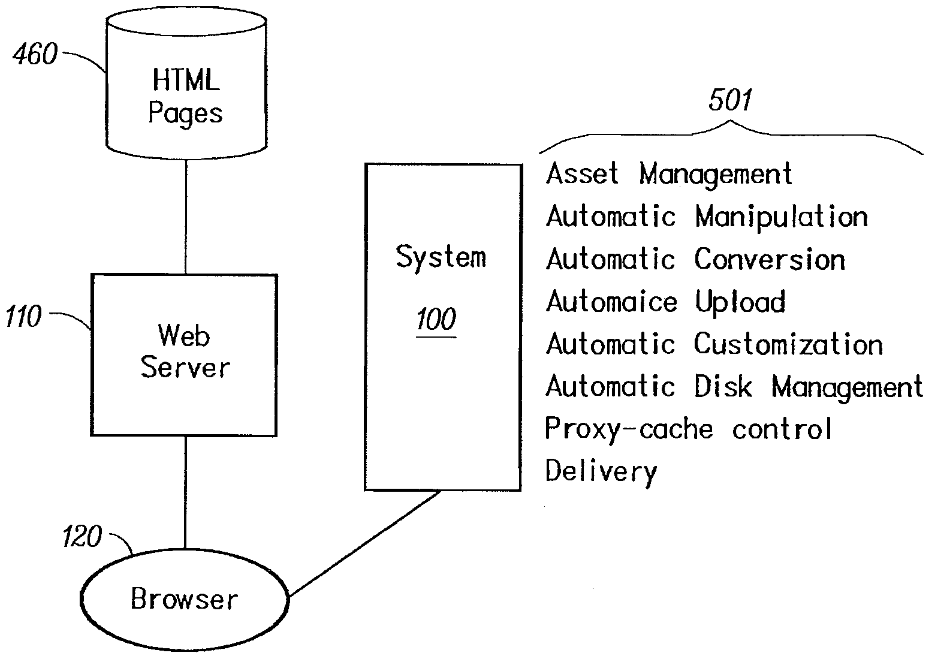

460
HTML Pages
110
Web Server
120
Browser
System
100
501
Asset Management
Automatic Manipulation
Automatic Conversion
Automaice Upload
Automatic Customization
Automatic Disk Management
Proxy-cache control
Delivery

FIG. 19

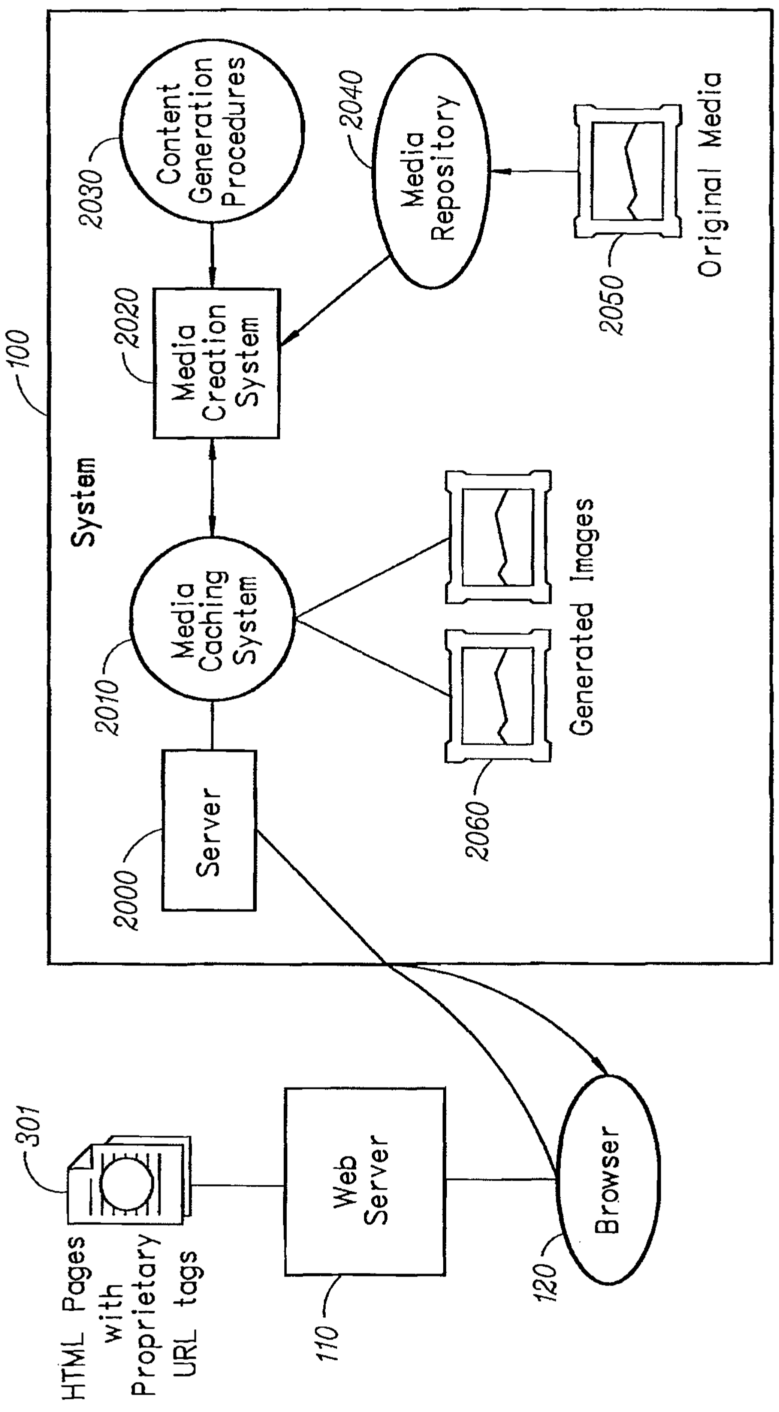

100
System
Server
2000
Media Caching System
2010
Media Creation System
2020
Content Generation Procedures
2030
Media Repository
2040
Original Media
2050
Generated Images
2060
HTML Pages with Proprietary URL tags
301
Web Server
110
Browser
120

FIG. 20

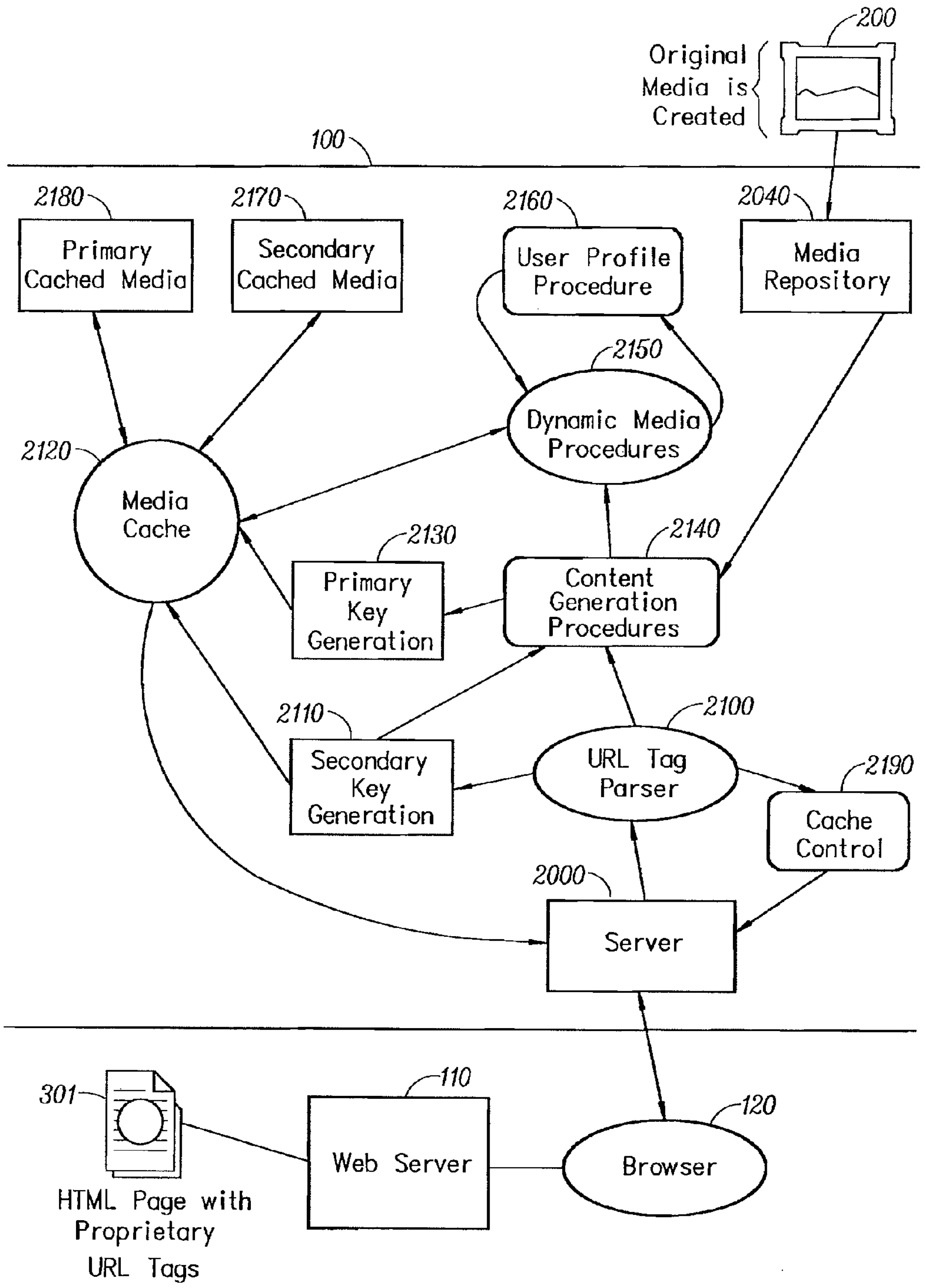

200
Original Media is Created
100
2180
Primary Cached Media
2170
Secondary Cached Media
2160
User Profile Procedure
2040
Media Repository
2150
Dynamic Media Procedures
2120
Media Cache
2130
Primary Key Generation
2140
Content Generation Procedures
2110
Secondary Key Generation
2100
URL Tag Parser
2190
Cache Control
2000
Server
301
HTML Page with Proprietary URL Tags
110
Web Server
120
Browser

FIG. 21

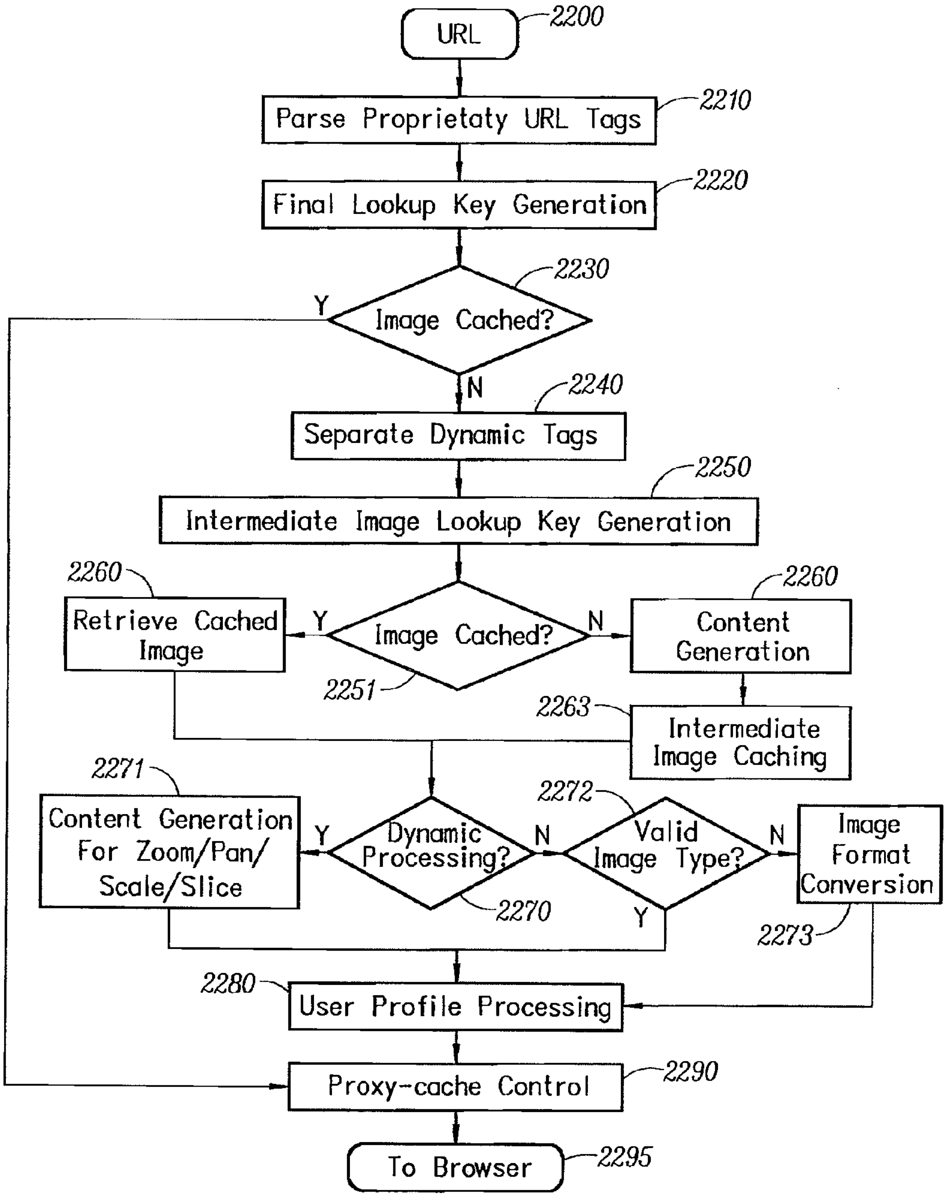

URL
2200
Parse Proprietaty URL Tags
2210
Final Lookup Key Generation
2220
Image Cached?
2230
Y
N
Separate Dynamic Tags
2240
Intermediate Image Lookup Key Generation
2250
2260
Retrieve Cached Image
Y
Image Cached?
N
2251
2260
Content Generation
2263
Intermediate Image Caching
2271
Content Generation For Zoom/Pan/ Scale/Slice
Y
Dynamic Processing?
N
2270
2272
Valid Image Type?
N
Y
Image Format Conversion
2273
2280
User Profile Processing
Proxy-cache Control
2290
To Browser
2295

FIG. 22

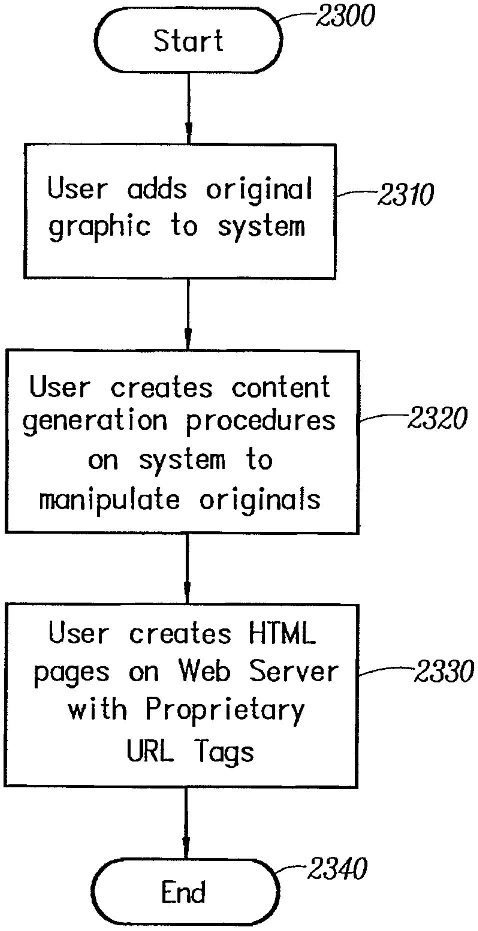

Start
2300
User adds original graphic to system
2310
User creates content generation procedures on system to manipulate originals
2320
User creates HTML pages on Web Server with Proprietary URL Tags
2330
End
2340

FIG. 23

AUTOMATED MEDIA DELIVERY SYSTEM

CROSS REFERENCE TO RELATED APPLICATIONS

This application is a Divisional of U.S. Ser. No. 12/173,747, filed Jul. 15, 2008, which is a Divisional of U.S. Ser. No. 11/269,916, filed Nov. 7, 2005 now abandoned, which is a Continuation-in-Part of U.S. Ser. No. 09/929,904, filed Aug. 14, 2001, now U.S. Pat. No. 6,964,009 granted on Nov. 8, 2005, which is a Continuation-in-Part of U.S. Ser. No. 09/425,326, filed Oct. 21, 1999, now U.S. Pat. No. 6,792,575, granted on Sep. 14, 2004, each of which is hereby incorporated in its entirety by this reference thereto.

BACKGROUND OF THE INVENTION

1. Technical Field

The invention relates to software systems. More particularly, the invention relates to an Internet server-based software system that provides delivery of automated graphics and other media to Web sites for access by an end user or consumer.

2. Description of the Prior Art

Most Web sites today are primarily handmade. From the guy publishing a simple online technology newsletter from his home, to the Fortune 1000 company's multi-tiered site with hundreds of pages of text, images, and animations, the Web developer and each of his HTML-coding and graphics-producing coworkers toil page by page and image by image. Thousands of established online companies employ hundreds of highly-skilled workers just to produce and maintain their Web sites. After all, the Web is now a major selling vehicle and marketing medium for many of these companies. The Web has even sprouted service industries such as, for example, public companies with multi-billion dollar valuations created just to consult and produce Web sites for others.

Most Web developers who use established WYSIWYG tools in the industry still must produce each page on their Web site one by one. The same rate applies to preparing and placing images, animations, and other visual assets. Each page represents its own set of issues ranging from whether to use GIF, JPEG, or PNG file formats, to finding the optimum bit depth for each image to ensure the fastest downloading through the different browsers of the consumer. The bottlenecked state of the customer's workflow to produce graphics for Web pages can be described as follows:

Current Workflow for Creating Web Graphics
Original Artwork/Asset Creation
  Use third-party point products
Asset Editing
  Scale/reduce/slice
Asset Format Conversion
  JPEG/GIF/PNG
Asset Staging
  Place in Web file system
  Edit HTML
Create/Modify HTML for particular page
Store HTML on Web server
View final pages
Repeat process for each version of each graphic on each page
Estimated time
  Two hours per page times the number of pages

Also, from a user's perspective, the current state of the art is to offer the consumer zooming and panning capabilities so that by clicking on an image the consumer can view more closely or from a different angle. On the horizon are pages with three-dimensional imagery that enable a user to move around a page that can look more like a room than a brochure. While interesting, these features are merely incremental improvements to a consumer's surfing experience.

D. C. A. Bulterman, *Models, Media, and Motion: Using the Web to Support Multimedia Documents*, Proceedings of 1997 International Conference on Multimedia Modeling, Singapore, 17-20 Nov. 1997 discloses "an effort underway by members of industry, research centers and user groups to define a standard document format that can be used in conjunction with time-based transport protocols over the Internet and intranets to support rich multimedia presentations. The paper outlines the goals of the W3C's Synchronized Multimedia working group and presents an initial description of the first version of the proposed multimedia document model and format."

Text and Graphics on UMI's Pro Quest Direct: The Best (yet) *of both Worlds, Online*, vol. 21, no. 2, pp. 73-7, March-April 1997 discloses an information system that offers "periodical and newspaper content covering a wide range of business, news, and professional topics . . . letting the user search both text and graphics and build the product to suit. Articles can be retrieved in varying levels of detail: citation, abstracts, full text, and text with graphics. Images come in two flavors: Page Image, a virtual photocopy, and Text+Graphics, in which graphics are stored separately from the text and are manipulable as discrete items . . . . [The system] comes in two versions: Windows and Web."

John Mills Dudley, *Network-Based Classified Information Systems*, AU-A-53031/98 (27/08/98) discloses a "system for automatically creating databases containing industry, service, product and subject classification data, contact data, geographic location data (CCG-data) and links to web pages from HTML, XML, or SGML encoded web pages posted on computer networks such as Internets or Intranets . . . . The . . . databases may be searched for references (URLs) to web pages by use of enquiries which reference one or more of the items of the CCG-data. Alternatively, enquiries referencing the CCG-data in the databases may supply contact data without web page references. Data duplication and coordination is reduced by including in the web page CCG-data display controls which are used by web browsers to format for display the same data that is used to automatically update the databases."

Cordell et al, Automatic Data Display Formatting with A Networking Application, U.S. Pat. No. 5,845,084 (Dec. 1, 1998) discloses a placeholder image mechanism. "When a data request is made, the data transfer rate is monitored. When the receive data transfer rate is slow, and the data contains an embedded graphical image of unknown dimensions, a small placeholder image is automatically displayed for the user instead of the actual data. The small placeholder image holds a place on a display device for the data or the embedded graphical image until the data or embedded graphical image is received. When embedded graphical image is received, the placeholder image is removed, and the display device is reformatted to display the embedded graphical image."

Jonathon R. T. Lewis, System For Substituting Tags For Non-Editable Data Sets In Hypertext Documents And Updating Web Files Containing Links Between Data Sets Corresponding To Changes Made To The Tags, U.S. Pat. No. 5,355,472 (Oct. 11, 1994) discloses a "hypertext data processing system wherein data sets participating in the hypertext document may be edited, the data processing system inserting tags into the data sets at locations corresponding to the hypertext links to create a file which is editable by an editor and the data processing system removing the tags, generating a revised data set and updating the link information after the editing process. Its main purpose is to preserve the linking hierarchy that may get lost when the individual data sets get modified."

Wistendahl et al, System for Mapping Hot Spots in Media Content Interactive Digital Media Program, U.S. Pat. No. 5,708,845 (Jan. 13, 1998) discloses a "system for allowing media content to be used in an interactive digital media (IDM) program [that] has Frame Data for the media content and object mapping data (N Data) representing the frame addresses and display location coordinates for objects appearing in the media content. The N Data are maintained separately from the Frame Data for the media content, so that the media content can be kept intact without embedded codes and can be played back on any system. The IDM program has established linkages connecting the objects mapped by the N Data to other functions to be performed in conjunction with display of the media content. Selection of an object appearing in the media content with a pointer results in initiation of the interactive function. A broad base of existing non-interactive media content, such as movies, videos, advertising, and television programming can be converted to interactive digital media use. An authoring system for creating IDM programs has an object outlining tool and an object motion tracking tool for facilitating the generation of N Data. In a data storage disk, the Frame Data and the N Data are stored on separate sectors. In a network system, the object mapping data and IDM program are downloaded to a subscriber terminal and used in conjunction with presentation of the media content."

Rogers et al, Method for Fulfilling Requests of A Web Browser, U.S. Pat. No. 5,701,451 (Dec. 23, 1997) and Lagarde et al, Method for Distributed Task Fulfillment of Web Browser Requests, U.S. Pat. No. 5,710,918 (Jan. 20, 1998) disclose essentially "improvements which achieve a means for accepting Web client requests for information, obtaining data from one or more databases which may be located on multiple platforms at different physical locations on an Internet or on the Internet, processing that data into meaningful information, and presenting that information to the Web client in a text or graphics display at a location specified by the request."

Tyan et al, HTML Generator, European Patent Application No. EP 0843276 (May 20, 1998) discloses "generating an HTML file based on an input bitmap image, and is particularly directed to automatic generation of an HTML file, based on a scanned-in document image, with the HTML file in turn being used to generate a Web page that accurately reproduces the layout of the original input bitmap image."

TrueSpectra has a patent pending for the technology employed in its two products, IrisAccelerate and IrisTransactive. These products are designed for zooming and panning and simple image transformations and conversions, respectively. They support 10 file formats and allow developers to add new file formats via their SDK. They do not require the use of Flashpix for images. However, their documentation points out that performance is dependent on the Flashpix format. The system would be very slow if a non-Flashpix format was used.

TrueSpectra allows the image quality and compression to be set for JPEGs only. The compression setting is set on the server and all images are delivered at the same setting.

TrueSpectra has a simple caching mechanism. Images in the cache can be cleared out automatically at certain times and it does not have any dependency features for image propagation. The Web server needs to be brought down in order to update any original assets.

TrueSpectra does not require plug-ins to operate features such as zooming/panning or compositing. The alternative to plug-ins is using their Javascript or active server page technology. These technologies are used by many Web sites to provide interactivity, but not all Web browsers work correctly with these technologies.

TrueSpectra relies on Flashpix as its native file format and does not support media types such as multi-GIFs and sound formats. Flashpix files are typically larger than most file formats. Access to files is faster for zooming and panning, but appears to be quite slow.

The key to IrisTransactive is the compositing subsystem. It requires three things to build a shopping solution using image composition.

1) The original images must be created. It is suggested that the image be converted to Flashpix for better performance.
2) All of the individual images must be described in XML using the image composer program. The program allows the editor to specify anchor points, layer attributes, and layer names. The resulting file is between 5 k and 50 k.
3) The Web designer must place HTML referring to the XML in the Web site. By specifying parameters to the XML, the Web designer can turn on or off layers.

The herein above process for compositing images enables Web designers to create shopping sites. However, a lot of overhead is the result. The XML documents add 5 k-50 k to a Web site. The compositing commands that are embedded in the HTML are difficult to understand. And, because the compositing feature requires several steps to implement, it is not suitable for every image on a Web site. The process seems to be designed for the specific purpose of shopping.

MediaBin™ is limited to activities behind the firewall automating only the "post-creative busywork." In addition, MediaBin requires the use of an application server to function through a web interface. Thus images may not be directly added to any existing web page.

Macromedia's Generator operates by embedding variables in their proprietary Flash format. Therefore the actual imaging operations are somewhat limited and cannot be controlled directly from a web page request.

MGI Software sells point solutions that require end-users to download a viewer to process a proprietary image format.

PictureIQ offers a server-side image-processing appliance that provides a limited set of Photoshop functionalities. This appliance runs on the web-page server, processes information embedded in the web page, and rewrites the web page with image data.

The disclosed prior art fail to provide systems and methodologies that result in a quantum leap in the speed with which they can modify and add images, video, and sound to sites, in the volume of data they can publish internally and externally, and in the quality of the output. The development of such an automated media delivery system would constitute a major technological advance.

It would be advantageous to empower an end user with flexibility and control by providing interactive page capabilities.

It would be advantageous from an end user's perspective to generate Web pages that contain active graphics. For example, clicking on a Corvette image will cause a simple menu to pop up suggesting alternative colors and sizes in which to see the car. Clicking on portions of the image, such as a fender, can call up a close-in view of the fender.

It would be advantageous to provide an automated graphics delivery system that becomes part of the Web site infrastructure and operates as part of the Web page transaction and that thereby provides a less expensive and less time-consuming process.

It would be advantageous to provide a system for automated processing and delivery of media (images, video, and sound) to a Web server whereby it eliminates the laborious post-production and conversion work that must be done before a media asset can be delivered on a Web server.

It would be advantageous to create a dynamic Web site, wherein images are generated on demand from original assets, wherein only the original assets need to be updated, and wherein updated changes propagate throughout the site.

It would be advantageous to provide a system that generates media based on current Web server traffic thereby optimizing throughput of the media through the Web server.

It would be advantageous to provide a system that generates media that is optimized for the Web client, wherein client connection speed determines optimum quality and file size.

It would be advantageous to provide a system that generates media, whereby the media is automatically uploaded.

It would be advantageous to provide a system that automatically caches generated media so identical requests can be handled without regeneration of images.

It would be advantageous to provide a system that resides behind the Web server, thereby eliminating security issues.

It would be advantageous to provide a system wherein the client browser does not require a plug-in.

It would be advantageous to provide a system wherein the system does not require any changes to a Web server.

It would be advantageous to provide a system wherein the system manages the Web server media cache.

It would be advantageous to provide a system wherein the Web media is generated only if requested by a client browser.

It would be advantageous for a system to reduce the need for a Web author to create different versions of a Web site, the system automatically handling image content.

It would be advantageous to provide dynamic imaging capabilities, have a more complete set of image processing functionality, and be controlled directly through an image URL.

It would be advantageous to provide an end-to-end solution requiring only a standard browser that is completely controllable using the proprietary tags contained within a simple image link in the web page.

It would be advantageous to run an image application as a separate server controlled directly by single image requests to that server, such that any web server, even one that is only sending static HTML can access imaging features.

SUMMARY OF THE INVENTION

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, generated media is cached such that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a schematic diagram showing the placement of the system within a current Web infrastructure according to the invention;

FIG. **2** is a schematic diagram showing how a typical Web site delivers an HTML document and its graphics to a Web browser according to the prior art;

FIG. **3** is a schematic diagram showing delivery of an HTML document and media to a Web browser according to the invention;

FIG. **4** is a schematic diagram showing the components involved in Web site administration according to the prior art;

FIG. **5** is a schematic diagram showing the components of the system involved in Web site administration according to the invention;

FIG. **6** is a simple overview showing the components of the system according to the invention;

FIG. **7** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to the invention;

FIG. **8** is a flow chart showing an authoring process according to the invention;

FIG. **9** is a flow chart showing an HTML parsing process according to the invention;

FIG. **10** is a flow chart showing a media creation process according to the invention;

FIG. **11** is a screen shot showing an administration tool according to the invention;

FIG. **12** displays a structure of a database record used for the system according to the invention;

FIG. **13** shows original media to be processed according to the invention;

FIG. **14** shows a portion on an HTML document with a proprietary tag according to the invention;

FIG. **15** shows an HTML document and an HTML document source according to the invention;

FIG. **16** shows a generated GIF image according to the invention;

FIG. **17** is a schematic diagram of an image system within a typical Web infrastructure according to the invention;

FIG. **18** is a schematic diagram showing delivery of an HTML document and original media according to the invention;

FIG. **19** is a schematic diagram showing components of Web site administration according to a preferred embodiment of the invention;

FIG. **20** is a simple overview showing components of the image system according to a preferred embodiment of the invention;

FIG. **21** is a schematic diagram showing process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention;

FIG. **22** shows a flowchart of a content generation procedure according to a preferred embodiment of the invention; and

FIG. **23** is a flow chart showing an authoring process according to a preferred embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, the generated media is cached so that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

A detailed description of such automatic media delivery system operating in parallel with existing Web site infrastructure is found below in the section under the heading as such.

FIG. **1** is a schematic diagram showing the placement of the system within a current Web infrastructure according to a preferred embodiment of the invention. The system **100** is attached to a Web server **110**, which is connected to multiple client browsers **120**(*a-d*) via the Internet **130**.

FIG. **2** is a schematic diagram showing how a typical Web site delivers an HTML document and its graphics to a Web browser according to the prior art. An original media **200** is passed to post-production systems **210**, wherein the media **200** is manipulated by hand and prepared for the Web. The result is a Web media **220**. The Web media **220** and an associated HTML document **230** referring to the media **220** by media tags are input to a Web server **110** for a Web browser **120** to view via the Internet **130**.

FIG. **3** is a schematic diagram showing delivery of an HTML document and media to a Web browser according to a preferred embodiment of the invention. An original media **200** and an HTML document embedded with proprietary media tags **300** are input into the system **100**. The system **100** generates a Web-safe media **220** and a modified HTML document **230** that refers to the Web media, and automatically loads them onto the Web server **110** for view by a Web browser **120** via the Internet **160**.

FIG. **4** is a schematic diagram showing components involved in Web site administration according to the prior art. Original media assets **400** are original images, video, or sound that have not been prepared for the Web. Web sites usually need to manage the placement of media on the network for easy retrieval by Web designers. Post-production systems **410** vary from Web site to Web site.

Post-production systems **410** are usually custom procedures that Web designers use to convert an original media, such as an image, to one that can be displayed on the Web. Post-production systems **410** also upload finished images to Web image systems. Web images **420** are Web versions of the original images. Web images **420** are ready for retrieval by the Web server **110** to be delivered to a Web browser **120**. Any image to be modified or updated must pass through the herein above three components before it can be delivered to the Web browser **120**. HTML pages **460** have references to Web images **420**.

FIG. **5** is a schematic diagram showing the components involved in Web site administration according to a preferred embodiment of the invention. Web site administration is simplified using the claimed invention. Asset management, automatic image manipulation, automatic image conversion, automatic image upload, and automatic disk management **500** are provided by the claimed invention.

FIG. **6** is a simple overview showing the components of the system according to a preferred embodiment of the invention. HTML with proprietary tags **300** is the original HTML document that is embedded with proprietary tags which describe how the images are to be manipulated for the Web. Java servlet engine **600** is a third-party product that allows the system **100** to interface with the Web server **110** and execute Java servlet code. The Web server **110** is third-party software that delivers Web pages to a Browser **120**. The Browser **120** views Web pages that are sent from the Web server **110**. Modified HTML with system created images **230** are a final result of the system. Modified HTML **230** is a standard HTML document without proprietary embedded tags and with standard Web graphics.

The System.

A preferred embodiment of the system **100** is provided.

HTML parsing subsystem **610** parses through an HTML document and searches for proprietary tags. If it finds a proprietary tag it hands it to a media caching subsystem **620** for further processing. The media caching subsystem **620** returns a standard HTML tag. The HTML parsing subsystem **610** then replaces the proprietary tag it found with the returned tag. The parsing subsystem **610** then continues searching for a next proprietary tag, repeating the process herein above. The process is finished when no more proprietary tags can be found.

The media caching subsystem **620** determines if an image has been created for the requested proprietary tag. If the image has already been created and the files that built that image have not been modified, the media caching subsystem **620** returns an HTML tag that refers to a previously-generated image. If the image has not been created, the media caching subsystem **620** hands the HTML tag to a media creation subsystem **630**. The media creation subsystem **630** returns an image to the media caching subsystem **620**. The media caching subsystem **620** adds the created image and the HTML tag to a media cache database **640**.

The media cache database **640** contains references to the created images **645**. In a preferred embodiment, the references are the script used to create the image, the names of the images used to create the image, the dates of those files, and the HTML that represents the created image. The media caching subsystem **620** performs lookups in this database to determine if the image has been created. If the image has not been created the media caching subsystem **620** calls upon the media creation subsystem **630** to create the image and then store the results in the media cache database **640**.

The media creation subsystem **630** takes a proprietary tag from the media caching subsystem **620** and generates an image. The image is generated by deciphering the tag and handing it to the media processing engine **650**. After the image is created, the media creation subsystem returns the name of the newly created image to the media caching subsystem **620**.

The media processing engine **650** interprets the proprietary tag and generates the image. The media processing engine **650** looks up images in a media repository to obtain the location of the original file.

The media repository **660** contains original images **665** used in the system **100**.

FIG. **7** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention. An original media **200** is created. The media **200** is placed into the system **100** in the media repository **660**. Similarly, an HTML document with proprietary tags **300** is created and placed on a Web server **110**. A user requests a Web page from a Web browser **120**. The Web server **110** passes the requested page to an HTML parser **610**. The HTML parser **610** parses HTML looking for media tags. The parser **610** looks up media tags in a media tags database **640**. If the media tag is found, then the system **100** produces a modified HTML document **230**. Otherwise, the media creation subsystem **630** uses the media tag to generate a Web media **220**. The generated Web media **220** is placed in a media cache subsystem **620**. The proprietary media tag is converted by a converter **700** to a standard HTML tag that refers to the generated media **220** in cache. The media tag and the HTML equivalent are stored in the media tags database **640**. Media tags are replaced by standard HTML equivalent to provide a modified HTML document **230**. The modified HTML document **230** is delivered to the Web server **110**. The Web server **100** delivers the modified HTML document **230** to the browser **120** via the Internet for a user to view.

FIG. **8** is a flow chart showing an authoring process according to a preferred embodiment of the invention. The process starts (**800**) when a user adds an original graphic to the system (**810**). The user then creates an HTML document that contains proprietary media tags (**820**). The user then places the HTML document on a Web server (**830**) and ends the authoring process (**840**).

FIG. **9** is a flow chart showing an HTML parsing process according to a preferred embodiment of the invention. The process starts (**900**) when a consumer requests a Web page (**910**). A Web server hands the request of the Web page to the system (**920**). The system parses the Web page (**930**). The system looks for a media tag (**940**). If found, the system retrieves the HTML equivalent of the media tag (**950**) and replaces the media tag with the HTML equivalent tag (**960**). The system continues parsing the Web page for tags (**970**) by returning to step (**940**). When no more tags are found, the system delivers the modified Web page to the Web server (**980**) and therein ends the process (**990**).

FIG. **10** is a flow chart showing a media creation process according to a preferred embodiment of the invention. The process starts (**1000**) when the system requests an HTML equivalent to a proprietary media tag (**1010**). The Media tag is combined with bandwidth information (**1020**). The subsystem checks if the media tag already exists in the media tag database (**1030**). If it does, the subsystem checks if any of the original assets used to create the media have been changed (**1040**). If not, then the subsystem retrieves the HTML equivalent tag from the database (**1050**) and returns the HTML equivalent tag to the requesting system (**1060**). If any of the original assets used to create the media have been changed (**1040**), then the subsystem removes the media tag entry from the media database (**1070**) and creates the media using the media tag (**1080**). The subsystem then stores the media in a media cache (**1090**). The subsystem generates the HTML referring to the generated media (**1100**) and places the media tag and the HTML equivalent in the media tag database (**1110**). The HTML equivalent is returned to the requesting system (**1060**) and the process stops (**1120**).

The differences between using HTML and the proprietary tags disclosed herein are noted. HTML allows Web designers to create Web page layouts. HTML offers some control of the images. HTML allows the Web designer to set the height and width of an image. However, all of the other image operations disclosed herein are supported by the claimed invention and are not supported by HTML.

Table A herein below provides the claimed proprietary tags according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

TABLE A

Tags
Generate image
<freerideimage> mediascript </freerideimage>
Generate a standard Web image.
Generate thumbnail image linked to full image
<freerideimagethumbnail> mediascript <xs=size ys=size / freerideimagethumbnail>
Generate a thumbnail of specified size and link it to the full size version.
Generate zoom and pan image
<freerideimagezoom> mediascript </freerideimagezoom>
Generate a zoomable/panable image.
Security
<freerideimagesecure> </freerideimagesecure>
Specifies that all images found between these tags are secured images and the system will determine access before generating.

Table B herein below provides the claimed script commands according to a preferred embodiment of the invention. Additional commands may be added as needed.

TABLE B

Media processing script commands
Add Noise
Noise__AddNoise([amount=<value 1..999>] [gaussian] [grayscale])
This command adds noise to the image.
Adjust HSB
AdjustHsb([hue @ <value ±255>] [saturation @ <value ±255>] [brightness @ <value ±255>])
This command allows the HSB of an image to be altered. This can be applied to images of all supported bit-depths.
Adjust RGB
AdjustRgb([brightness @ <value ±255>] [contrast @ <value ±255>] [red @ <value ±255>] [green @ <value ±255>] [blue @ <value ±255>] [noclip @ <true, false>] [invert @ <true, false>])
This command allows the contrast, brightness, and color balance of an image to be altered.
Blur
Blur(radius @ <value 0..30>)
This command applies a simple blur filter on the image.

TABLE B-continued

Media processing script commands
Blur Convolve
Blur__Blur( )
This command commands perform a simple 3×3 convolution for blurring.
Blur Convolve More
Blur__MoreBlur( )
This command commands perform a stronger 3×3 convolution for blurring.
Blur Gaussian
Blur__GaussianBlur([radius=<value 0.1..250>])
This command applies a Gaussian blur to the image.
Blur Motion
Blur__MotionBlur([distance=<value 1..250>] [angle=<degrees>])
This command applies motion blurring to the image using the specified distance and angle.
Brush Composite
Composite(source @ {<User-Defined Media Object name>} [x @ <pixel>] [y @ <pixel>] [onto] [opacity @ <value 0..255>] [color @ <color in hexadecimal>] [colorize @ <true, false>] [saturation @ <value 0..255>])
This command composites the specified “brush” (foreground) image onto the current “target” (background) image.
Colorize
Colorize(color @ <color in hexadecimal> [saturation @ <value 0..255>])
This command changes the hue of the pixels in the image to the specified color.
Convert
Convert(rtype @ <bit-depth> {dither @ <value 0..10>])
This command converts the image to the specified type/bit-depth.
Convolve
Convolve(Filter @ <filtername>)
This command applies a basic convolution filter to the image. In a user interface driven system, the filters could be stored in files and edited/created by the user.
Crop/Resize Canvas
Crop([xs @ {<pixels>, <percentage + “%”>}] [ys @ {<pixels>, <percentage + “%”>}] [xo @ <left pixel>] [yo @ <top pixel>] [padcolor @ <color in hexadecimal>] [padindex @ <value 0..255>])
This command crops the media to a specified size.
Discard
Discard( )
This command removes the designated Media Object from memory.
Drop Shadow
DropShadow([dx @ <pixels>] [dy @ <pixels>] [color @ <color in hexadecimal>] [opacity @ <value 0..255>] [blur @ <value 0..30>] [enlarge @ <true, false>])
This command adds a drop shadow to the image based on its alpha channel.
Equal
Equal(source @ {<User-Defined Media Object name>})
This command compares the current media with the one specified. If the media are different in any way, an error value is returned.
Equalize
Equalize([brightness @ <−1, 0..20>] [saturation @ <−1, 0..20>])
This command equalizes the relevant components of the media. Equalization takes the used range of a component and expands it to fill the available range.
Export Channel
ExportGun(Channel @ <channelname>)
This command exports a single channel of the source as a grayscale image.
Find Edges
Stylize__FindEdges([threshold=<value 0..255>] [grayscale] [mono] [invert])
This command finds the edges of the image based on the specified threshold value.
Fix Alpha
FixAlpha( )
This command adjusts the RGB components of an image relative to its alpha channel.
Flip
Flip(<horizontal, vertical> @ <true, false>)
This command flips the media vertically or horizontally.
Frame Add
FrameAdd(Source @ <filename>)
This command adds the given frame(s) to the specified Media Object.
Glow/Halo
Glow(Size @ <value 0..30> [halo @ <value 0..size>] [color @ <color in hexadecimal>] [opacity @ <value 0..255>] [blur @ <value 0..30>] [enlarge @ <true, false>])
This command produces a glow or halo around the image based on the image’s alpha.
High Pass
Other__HighPass([radius=<value 0.1..250>])
This command replaces each pixel with the difference between the original pixel and a Gaussian blurred version of the image.
Import Channel
ImportGun(channel @ <channel name> source @ {<User-Defined Media Object name>} [rtype @ <bit-depth>])
This command imports the specified source image (treated as a grayscale) and replaces the selected channel in the original.
Load

TABLE B-continued

Media processing script commands
Load(Name @ <filename> [type @ <typename>] [transform @ <true, false>])
This command loads a media from the specified file.
Maximum
Other_Maximum([radius=<value 1..10>])
This command scans the area specified by the radius surrounding each pixel, and then replaces the pixel with the brightest pixel found.
Minimum
Other_Minimum([radius=<value 1..10>])
This command scans the area specified by the radius surrounding each pixel, and then replaces the pixel with the darkest pixel found.
Normalize
Normalize([clip @ <value 0..20>])
This command expands the volume of the sample to the maximum possible.
Pixellate Mosaic
Pixellate_Mosaic([size=<value 2..64>])
This command converts the image to squares of the specified size, where each square contains the average color for that part of the image.
Pixellate Fragment
Pixellate_Fragment([radius=<value 1..16>])
This command produces four copies of the image displaced in each direction (up, down, left, right) by the specified radius distance and then averages them together.
Quad Warp
QuadWarp([tlx=<position>] [tly=<position>] [trx=<position>] [try=<position>] [blx=<position>] [bly=<position>] [brx=<position>] [bry=<position>] [smooth])
This command takes the corners of the source image and moves them to the specified locations, producing a warped effect on the image.
Reduce to Palette
Reduce([colors @ <num colors>] [netscape @ <true, false>] [b&w @ <true, false>] [dither @ <value 0..10>] [dithertop @ <value 0..10>] [notbackcolor] [pad @ <true, false>])
This command applies a specified or generated palette to the image.
Rotate
Rotate(Angle @ <value 0..359> [smooth @ <true, false>] [enlarge @ <true, false>] [xs @ <pixels>]
[ys @ <pixels>])
This command rotates the media by the specified angle in degrees.
Rotate 3D
Rotate3d([anglex @ <angle ±89>] [angley @ <angle ±89>] [distance @ <value>])
This command rotates the image in 3D about either the x-axis or y-axis.
Save
Save([type @ <image-type>])
This command saves a media to the specified file.
Scale
Scale([xs @ {<pixels>, <percentage + "%">}] [ys @ {<pixels>, <percentage + "%">}] [constrain @ <true, false>] [alg @ {"fast", "smooth", "outline"}] [x1 @ <pixels>] [y1 @ <pixels>] [x2 @ <pixels>] [y2 @ <pixels>])
This command scales the image to the specified size.
Select
Selection([source @ <User-Defined media Object>}] [remove @ <true, false>] [invert @ <true, false>]
[backcolor] [color=<color>] [index=<value>] [opacity @ <value 0..255>])
This command manages the selected region for the current Media Object.
Set Color
SetColor([backcolor @ <color in hexadecimal>] [forecolor @ <color in hexadecimal>] [backindex @ <value 0..255>] [foreindex @ <value 0..255>] [transparency @ ("on","off")])
This command allows the background color, foreground color, and transparency state of an image to be set.
Set Resolution
SetResolution([dpi @ <value>] [xdpi @ <value>] [ydpi @ <value>])
This command changes the DPI of the image in memory.
Sharpen
Sharpen_Sharpen( )
This command sharpens the image by enhancing the high-frequency component of the image.
Sharpen More
Sharpen_SharpenMore( )
This command sharpens the image by enhancing the high-frequency component of the image, but is stronger than the standard sharpening.
Stylize Diffuse
Stylize_Diffuse([radius=<value 0..>] [lighten] [darken])
This command diffuses the image by randomizing the pixels within a given pixel radius.
Stylize Embose
Stylize_Emboss([height=<value 1..10>] [angle=<degrees>] [amount=<percentage 1..500>])
This command converts the image to an embossed version.
Text Drawing
DrawText(Text @ <string> Font @ <font file> [size @ <value>]
[color @ <color in hexadecimal>] [smooth @ <true, false>] [<left, right, top, bottom> @ <true, false>]
[x @ <pixel>] [y @ <pixel>] [wrap @ <pixel-width>] [justify @ {left,center,right}] [angle @ <angle>])

TABLE B-continued

Media processing script commands
This command composites the specified text string onto the image.
Text Making
MakeText(text @ <string> font @ <font file> [path @ <path to font directory>] [size @ <value 1..4095>]
[color @ <color in hexadecimal>] [smooth @ <true, false>] [wrap @ <pixel-width>]
[justify @ {left,center,right}] [angle @ <angle>])
This command creates a new image that includes only the specified text.
Trace Contour
Stylize__TraceContour([level=<value 0..255>] [upper] [invert])
This command traces the contour of the image at the specified level (for each gun).
Unsharpen Mask
Sharpen__UnsharpMask([amount=<percentage 1..500>] [radius=<value 0.1..250>]
[threshold=<value 0..255>])
This command enhances the edges and detail of an image by exaggerating differences between the image and a gaussian blurred version of the same image.
Zoom
Zoom([xs @ <pixels>] [ys @ <pixels>] [scale @ <value>] [x @ <left pixel>] [y @ <top pixel>])
This command zooms in on a specified portion of the media and fits it to the specified size. This constitutes a crop followed by a scale.

Table C herein below provides a list of features provided by a preferred embodiment of the invention. It is noted that the list of features included in Table C is by no means complete. In other embodiments, the list of features is expanded or reduced as needed.

TABLE C

System Feature List
Reads and writes various file formats:
BMP, GIF, JPG, PNG, TIF, PICT, TGA, PSD, FPX;
Supports many image processing operations;
Dynamically creates Web images from original assets;
Dynamically creates thumbnail images;
Dynamically creates images that can be panned and zoomed without browser plug-ins or special file formats;
Automatically propagates changes of original assets throughout a Web site;
Uses an intelligent caching mechanism:
Clean up image cache on demand;
Eliminates orphaned image files; and
Optimizes Web server cache by providing most recent images;
Renders TrueType fonts on the server instead of browser;
Uses intelligent scaling of line drawings;
Allows Web designers to manipulate images with proprietary tags;
Preserves original image assets;
Optimizes Web server traffic by adjusting the bandwidth of graphics;
Optimizes images for client connection speed;
Allows clients to specify the quality of images on a Web site; and
Allows Web designers to dynamically create images by manipulating proprietary tags in their applications (server or client side).

FIG. **11** is a screen shot showing an administration tool according to a preferred embodiment of the invention. Specifically, FIG. **11** shows an administration page that contains cached images of generated scripts. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **12** displays a structure of a database record used for the system according to a preferred embodiment of the invention. A Script Table **1200** has 5 columns, Media Script **1210**, HTML Equivalent **1220**, Bandwidth **1230**, Generated File **1240**, and Dependency List **1250**. A Dependency Table **1260** has two columns, File Name **1270** and Modification Date **1280**.

Snowboard Store Example.

Background.

The snowboard store highlights several features of the claimed system. The snowboard store is an imaginary store that allows a user to configure his or her snowboard. The store consists of five logos, five board colors, and four boards. The consumer clicks on the buttons to change the snowboard represented in the middle of the screen. When the consumér has configured the snowboard they the snowboard can be purchased by selecting a buy button.

Prior Art Method.

To create the snowboard site today, the Web designer must render all possible combinations of the board. The number of combinations is five logos×five board colors×four boards=100. The designer also must render all the buttons. The creation process is very tedious and involves a lot of production work. Typically, most Web sites do not even attempt such an endeavor. Also, other issues must be addressed, such as, for example, updating the Web site and scripting. For example, updating a single logo involves updating a minimum of 20 images.

The prior art method sustains a graphic intensive site that requires management of at least 100 images. Updates to the Web site are time-consuming and prone to human error.

The Claimed Method.

A preferred embodiment of the method scripts the image creation process in HTML to create a dynamic Web site. There is no need to create over 100 images. The claimed system generates images on demand. The Web site only needs to create original assets. The scripting process involves writing the proprietary scripts. In the current example herein, scripting buttons is very simple. Once one button is created, simply copy and paste the HTML to create another button or many buttons. Only the name of the image to be overlaid on the button must be changed. The Webmaster then creates a simple program that reads what object a user has clicked on and generates a proprietary tag. The tag is then sent to the claimed system to generate a center image.

The claimed method allows the creation of all 100 combinations automatically. When the Web site receives an updated image, only the original image needs to be updated. Any change to the original image automatically propagates throughout the system. The Web site is easier to manage. Testing of the Web site is easier because there is no need to test all 100 combinations. A small subset of combinations will guarantee adequate coverage.

Processing of an Image Tag Example (FIG. **13-16**).

FIG. **13** shows two original images **1300** and **1310** to be processed according to a preferred embodiment of the invention.

FIG. **14** shows a portion on an HTML document with a proprietary tag **1400**, <freerideimage></freerideimage> according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **15** shows an HTML document **1500** as viewed in a browser and an HTML document source **1510**, according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **16** shows a generated GIF image **1600** according to a preferred embodiment of the invention.

Automatic Media Delivery System Operating in Parallel with Existing Web Site Infrastructure.

It should be noted that the words, media, graphics, and images are used herein interchangeably.

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, the generated media is cached so that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

A preferred embodiment of the invention is described with reference to FIG. **17**. FIG. **17** is a schematic diagram of an image system within a typical Web infrastructure according to the invention. The image system **100** is placed in parallel to an existing Web server **110**. The image system **100** may be on-site or off-site to the Web infrastructure. Multiple client browsers **120***a*-**120***d* communicate with both the Web server **110** and the image server **100** via the Internet **130**.

The delivery of an HTML document and media according to a preferred embodiment of the invention is described with reference to FIG. **18**. Resource locators (URLs) are placed within HTML documents **301** accessible to the Web server **110**. These URLs direct browsers to generate requests for media to the system **100**. The system processes such URLs by interpreting the proprietary tags, executing the indicated image generation procedures on the original media **200**, and returning derivative Web-safe media to client browsers **120***a*-**120***d* via the Internet **130**. Additionally, such generated media is cached on the image server **100** and, therefore need not be regenerated for subsequent requests.

Web site administration according to another preferred embodiment of the invention is described with reference to FIG. **19**. FIG. **19** is a schematic diagram showing the components of Web site administration according to a preferred embodiment of the invention, whereby Web site administration is simplified. The preferred embodiment provides, but is not limited to the following services: asset management, automatic image manipulation, automatic image conversion, automatic image upload, automatic image customization based on browser characteristics, automatic disk management, automatic control of proxy caching, and image delivery **501**.

FIG. **20** is a simple overview showing components of the system according to a preferred embodiment of the invention. HTML pages with proprietary URL tags **301** describe how referenced media therein is to be manipulated for Web. Browsers **120** send such tags to the image system **100** as media requests. A server **2000** within the image system **100** receives the media requests, decodes the URL tags, and retrieves any media that already exists in the media caching system **2010**. Non-existent media is subsequently generated by a media creation system **2020** using original media **2050** stored in a media repository **2040** and using content generation procedures **2030**.

The Image System.

Following is a detailed description of the preferred embodiment of the invention with reference to FIG. **21** below.

The system receives a request for media through a URL containing proprietary tags for controlling image generation. The system parses this URL to determine the content generation procedure to execute, input to the content generation procedure, post-processing directives for, for example, zoom/pan/slice, browser properties, and any cache control directives. Such data is handed to a media caching subsystem that returns the requested image if found. If the image is not found, the information is handed to the media generation subsystem that executes the specified content generation procedure to produce a derivative image. The media generation subsystem returns one or more images to the media caching system for subsequent reuse.

The media caching subsystem is a mechanism for associating final or intermediate derivative media with the procedure, input, and user characteristics used to generate said media, specified through proprietary tags within the requested URL. This system may be implemented using a database, file system, or any other mechanism having capability to track such associations.

The media generation subsystem executes a primary content generation procedure to produce a derivative image whose identifier is provided to the media caching subsystem. This derivative image is composed of one or more original images acquired from the media repository. This media is then passed to the dynamic image content system, if necessary, to generate a subsequent derivative media suitably modified for the needs of zooming, panning, or slice. The resulting media is passed to the user profile system where it is again modified to account for any specific user browser characteristics specified using the proprietary URL tags. This media is then returned to the browser, along with any cache control directives encoded within the URL, and its identifier is passed to the media caching system for subsequent retrieval.

The dynamic content system operates on intermediate derivative images to generate image subsets or scalings used by Web site designers to implement zooming in on an image, panning across an image, slicing an image into parts, and the like for special Web page effects. The input to this system is cached by the media caching system such that the intermediate image need not be regenerated.

The user profile system operates on the final image about to be returned to the browser and may modify the image to account for individual needs of Web site users. The designer of a site is able to implement freely custom post-processing of images to meet the specific needs of their clients.

FIG. **21** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention. Original media **200** is created and placed into the system **100** in a media repository **2040**. A content generation procedure **2140** is created with instructions on how the media is to be transformed to create the desired Web page content. An HTML page **301** is created for the Web site comprising the system **100**, the page containing one or more URLs directing a browser **120** to request the specified content generation procedure **2140** from the system **100** using input parameters specified with proprietary tags encoded within the URL. The browser **120** requests the Web page **301** from the Web site **110**. Upon receipt of the page **301**, the browser contacts the system **100** requesting media specified in the URL. The system parses the URL **2100** to determine the content generation procedure **2140** to execute, any corresponding input parameters to be used by such procedure, any dynamic content processing **2150** to be performed by dynamic media procedures, any user profile information **2160** to be used to modify the resulting image, and any cache control HTTP headers **2190** the site instructs to accompany the resulting image.

The parser generates a unique primary lookup key **2110** for the specified resulting media. If the key corresponds to an existing generated media **2180**, such media is returned immediately to the browser **120** through a media cache **2120**, and the transaction is complete. Otherwise, a media generation occurs. In the case of media generation requiring dynamic content processing, a unique secondary lookup key corresponding to intermediate media is generated **2130**. If intermediate media **2170** corresponding to this key is found, such media is passed directly to the dynamic media content system **2150** having dynamic media procedures, wherein appropriate action is taken to generate the required derivative from the intermediate media data. A unique key is generated for the derivative **2130** and passed to the media caching system **2120**. If the media caching system finds no such intermediate image, such intermediate image is generated according to instructions specified by the content generation procedure, cached by the media cache system **2120** as a secondary cached media **2170**, and passed to the dynamic media system **2150**. Again, appropriate action is taken to generate the required derivative from the intermediate image data.

The resulting image after any dynamic media processing is complete, is checked to ensure that the image is in a valid Web image format. If not, the image is automatically converted into a valid format.

The final media is passed to a user profile system **2160** wherein browser characteristics specified through proprietary tags within the URL are inspected, and appropriate modification to the media is performed, based on such characteristics. The resulting image is handed to the media cache system **2120** for caching and returned to the browser **120**.

FIG. **22** shows a flowchart of the content generation procedure according to a preferred embodiment of the invention. A URL containing proprietary tags (**2200**) is parsed (**2210**) to determine the content generation procedure to execute, any dynamic modifications to the media, user profile characteristics, and proxy-cache control. A unique final lookup key is generated for the media (**2220**) and the media cache is checked (**2230**). If the indicated media exists, control passes to proxy-cache control (**2290**) and the media is delivered to the browser (**2295**). Otherwise, dynamic media system tags are separated from content generation control tags (**2240**) and a unique intermediate image lookup key is generated (**2250**). The cache is then checked for such intermediate media (**2261**). If such intermediate media is found, it is used directly

for dynamic processing, if required. Otherwise, content is generated (**2262**) and cached (**2263**), and the result is evaluated for dynamic processing (**2270**). If dynamic processing is required, the media is operated upon by the dynamic content generator (**2271**), otherwise it is evaluated for valid content type (**2272**). If the content type is invalid, the media is automatically converted to a valid type (**2273**). The resulting image is then customized by the user profiling system (**2280**) for specified browser or client attributes. Finally, any cache-control directives specified are attached to the response (**2290**) and the media is delivered to the browser (**2295**).

FIG. **23** is a flow chart showing an authoring process according to a preferred embodiment of the invention. The process starts (**2300**) when a user adds an original graphic or other media (**2310**) to the system. The author then creates a content generation procedure (**2320**) containing instruction on how the original media should be processed to generate the desired Web page content. The user then creates an HTML document (**2330**) that refers to that image by using a URL pointing to a content generation procedure on the image server. The system ends (**2340**). The authoring subsystem assists the Web site designer with choosing parameters and with designing the content generation procedure such that the desired Web site graphic is obtained.

It should be appreciated that differences exist between specifying an image with a URL and requesting an image using a content creation process that interprets proprietary parameters encoded within a URL. That is, URLs allow Web site designers to load specific graphic images into a Web page. In contrast and according to the invention, URLs containing proprietary content creation tags initiate a process whereby graphic images for a site are automatically produced.

Table D below is a list of example proprietary URL tags used for content generation within the system according to the preferred embodiment of the invention. Additional tags may be added to the system as necessary.

TABLE D

Tags
f = function Names the content creation procedure used to generate all or part of the desired graphic.
args = arguments Supplies page dependent parameters used to control the content creation procedure from within the Web page.
cr = crop rectangle Specifies that portion of the image generated by the content generation procedure to be returned to the browser.
st = slice table Specifies a rectangular grid to be placed over the image produced by the content generation procedure, each portion of which can be returned to the browser.
sp = slice position Specifies that portion of the slice table grid placed over the image generated by the content creation procedure to be returned to the browser.
is = image size parameter Specifies scale factors to be applied to any portion of an image generated by any combination of a content generation procedure, arguments, crop rectangles, slice tables, and slice positions.
p = user profile string Specifies a user profile identifier used to modify the final image prior to returning the image to the browser, thus allowing clients to modify the image returned to the browser to account for individual browsing conditions.
c = cache control Specifies a proxy-cache control string to accompany the returned image within an HTTP header.

Table E below is a list of example supported content creation commands according to a preferred embodiment of the invention. Additional commands may be added as necessary.

TABLE E

Content Creation Commands	
Adjust HSB	Allows the HSB of an image to be altered.
Adjust RGB	Allows the contrast, brightness, and color balance of an image to be altered.
Colorize	Alters the hue of the pixels in the image to that of the specified color.
Brush Composite	Composites the specified brush image onto the current target image.
Convert	Converts the rasters to the specified type/bit-depth.
Crop	Crops the media to the specified size.
Dropshadow	Adds a drop shadow to the image, based on the alpha-channel.
Equalize	Performs an equalization on the relevant components of the media.
FixAlpha	Adjusts the RGB components of an image relative to its alpha-channel.
Flip	Flips the media vertically or horizontally.
Glow	Produces a glow or halo around the image.
Load	Loads in a media from the specified file.
Normalize	Similar to equalize, but for audio.
Reduce	Reduces the image to a specified palette.
Rotate	Rotates the media clockwise by the specified angle in degrees.
Save	Saves the media to the specified file.
Scale	Scales the media to the specified size.
SetColor	Allows the background color, foreground color, and transparency state of the media to be set.
Text Drawing	Composites the specified text onto the image.
Text Making	This command, instead of compositing text onto the target, creates a new image that just encloses the text.
Zoom	Zooms in on a specified portion of the media, and fits it to the specified size. Effectively this constitutes a crop followed by a scale.

Table F below lists comprises some, and is not limited to all major features of a preferred embodiment of the invention. Additional features may be added as necessary.

TABLE F

System Features
Reads and writes various file formats;
Supports many image processing operations;
Dynamically creates Web images from original assets;
Dynamically creates thumbnail images;
Dynamically and efficiently creates images that can be panned, zoomed, or sliced from original assets without Browser plugins;
Automatically propagates changes in original assets throughout the Web site;
Uses an intelligent caching mechanism for both final and intermediate graphics, comprising:
Clean up cache on demand;
Eliminates orphaned Web files; and
Optimizes Web server cache by providing most recent images;
Renders True-Type fonts on server instead of browser;
Uses intelligent scaling of line drawings;
Allows Web designers to manipulate images using a combination of content generation procedures and proprietary URL tags;
Preserves original image assets;
Optimizes Web server traffic by adjusting the bandwidth of graphics;
Optimizes images for client connection speed;
Allows clients to specify the quality of images on a Web site;
Allow site-specific customized image optimizations for a variety of purposes; and
Allows Web designers to dynamically create images by manipulating proprietary URL tags in applications.

Accordingly, although the invention has been described in detail with reference to a particular preferred embodiment, persons possessing ordinary skill in the art to which this invention pertains will appreciate that various modifications and enhancements may be made without departing from the spirit and scope of the claims that follow.

The invention claimed is:

1. A non-transitory computer program product for causing a computing system to dynamically transcode media content to be presented on a client presentation system, the computer program product comprising:

one or more computer-readable media, the one or more computer-readable media having stored thereon computer-executable instructions that, when executed by one or more processors of the computing system, cause the computing system to perform the following:

an act of receiving a request for media content to be delivered to a client presentation system, wherein the requested media content has a limited number of base transcoding profiles associated therewith, each base transcoding profile corresponding to a cached version of the requested media content;

at the time of the request, and without input by a network administrator, an act of automatically identifying transcoding parameters to be applied to the requested media content prior to delivery to the client presentation system, wherein identification of transcoding parameters is based on one or more formats of any client presentation system;

an act of transcoding the requested media content in accordance with the identified transcoding parameters, such that the identified transcoding parameters are used to perform additional incremental transcoding on top of the base transcoding profile;

wherein the act of act of transcoding the requested media content in accordance with the identified transcoding parameters comprises:

an act of selecting a pre-existing base transcoded version of the requested media content comprising intermediate derivative media that has been transcoded in accordance with only a portion of the identified transcoding parameters; and

an act of creating a final version by incrementally performing further transcoding of the pre-existing base transcoded version in accordance with a remaining portion of the identified transcoding parameters;

an act of causing the transcoded media content to be delivered to the client presentation system; and

an act of caching the transcoded media content.

2. A non-transitory computer program product in accordance with claim **1**, wherein the act of receiving a request for media content is performed by receiving the request over the Internet.

3. A non-transitory computer program product in accordance with claim **1**, wherein the client presentation system is a first client presentation system, the method further comprising:

an act of receiving a request for the media content to be delivered to a second client presentation system;

an act of determining that the transcoding parameters to be applied to the requested media content prior to delivery to the second client presentation system are the same as the transcoding parameters that were applied to the requested media content prior to delivery to the first client presentation system; and

an act of delivering the cached transcoded media content, rather than separately transcoding the requested media content again for the second client presentation system.

4. A non-transitory computer program product in accordance with claim **1**, wherein the requested media content is video media.

5. A non-transitory computer program product in accordance with claim **1**, wherein the identified transcoding parameters includes at least video size parameters.

6. A non-transitory computer program product in accordance with claim **1**, wherein the method further comprises:

prior to the act of transcoding, an act of determining that a transcoded representation of the requested media content transcoded using the identified transcoding parameters does not already exist.

7. A non-transitory computer program product in accordance with claim **1**, wherein the act of identifying transcoding parameters comprises an act of selecting a transcoding profile from a set of pre-existing transcoding profiles.

8. A non-transitory computer program product in accordance with claim **1**, wherein the act of identifying transcoding parameters comprises:

an act of reading one or more transcoding parameters from the request; and

an act of identifying the one or more transcoding parameters from the request as being the identified transcoding parameters to be applied.

9. A method for accessing dynamically transcoding media content, the method comprising:

an act of receiving a request for media content to be delivered to a client presentation system for media content, wherein the requested media content has a limited number of base transcoding profiles associated therewith, each base transcoding profile corresponding to a cached version of the requested media content;

at the time of the request, and without input by a network administrator, an act of automatically identifying transcoding parameters to be applied to the requested media content prior to delivery to the client presentation system, wherein identification of transcoding parameters is based on one or more formats of any client presentation system;

an act of determining that the transcoding parameters to be applied to the requested media content prior to delivery to the client presentation system are the same as transcoding parameters that are being applied to the requested media content prior to delivery to another client presentation system;

an act of transcoding the requested media content in accordance with the identified transcoding parameters, such that the identified transcoding parameters are used to perform additional incremental transcoding on top of the base transcoding profile;

wherein the act of act of transcoding the requested media content in accordance with the identified transcoding parameters comprises:

an act of selecting a pre-existing base transcoded version of the requested media content comprising intermediate derivative media that has been transcoded in accordance with only a portion of the identified transcoding parameters; and

an act of creating a final version by incrementally performing further transcoding of the pre-existing base transcoded version in accordance with a remaining portion of the identified transcoding parameters; and

an act of delivering the transcoded media content to both client presentation systems concurrently.

10. A non-transitory computer program product for causing a computing system to dynamically transcode media content to be presented on a client presentation system, the computer program product comprising:

one or more computer-readable physical and/or memory storage media, the one or more computer-readable memory media having stored thereon computer-executable instructions that, when executed by one or more processors of the computing system, cause the computing system to perform the following:

maintaining a transcoding profile library, the transcoding profile library comprising a plurality of transcoding profiles for media content;

an act of receiving a request for media content to be delivered to a client presentation system, wherein the requested media content has a limited number of base transcoding profiles associated therewith, each base transcoding profile corresponding to a cached version of the requested media content;

an act of identifying transcoding parameters to be applied to the requested media content prior to delivery to the client presentation system;

determining whether an existing base transcoding profile exists within the transcoding profile library corresponds to the requested media content transcoded for delivery to the client presentation system;

an act of transcoding the requested media content in accordance with the identified transcoding parameters, such that the identified transcoding parameters are used to perform additional incremental transcoding based on multiple transcoding parameters on top of a base transcoding profile;

wherein the act of act of transcoding the requested media content in accordance with the identified transcoding parameters comprises:

an act of selecting a pre-existing base transcoded version of the requested media content comprising intermediate derivative media that has been transcoded in accordance with only a portion of the identified transcoding parameters; and

an act of creating a final version by incrementally performing further transcoding of the pre-existing base transcoded version in accordance with a remaining portion of the identified transcoding parameters;

an act of delivering the transcoded media content to the client presentation system;

adding a new transcoding profile to the transcoding profile library corresponding to the identified transcoding parameters and the base transcoding profile; and

an act of caching the transcoded media content.

11. A computer program product in accordance with claim **10**, wherein the act of receiving a request for media content is performed by receiving the request over the Internet.

12. A computer program product in accordance with claim **10**, wherein the client presentation system is a first client presentation system, the method further comprising:

an act of receiving a request for the media content to be delivered to a second client presentation system;

an act of determining that the transcoding parameters to be applied to the requested media content prior to delivery to the second client presentation system are the same as the transcoding parameters that were applied to the requested media content prior to delivery to the first client presentation system; and

an act of delivering the cached transcoded media content, rather than separately transcoding the requested media content again for the second client presentation system.

13. A computer program product in accordance with claim **12**, wherein act of caching the transcoded media content comprises an act of first caching a transcoded stream of the transcoded media content while the act of transcoding is still occurring, and further caching a transcoded file of the transcoded media content after the act of transcoding has completed.

14. A computer program product in accordance with claim **13**, wherein the act of delivery the cached transcoded media content comprises:

an act of providing the transcoded stream to the second client presentation system.

15. A computer program product in accordance with claim **13**, wherein the act of delivery the cached transcoded media content comprises:

an act of providing the transcoded file to the second client presentation system.

16. A computer program product in accordance with claim **11**, wherein the requested media content is video media.

17. A computer program product in accordance with claim **11**, wherein the identified transcoding parameters includes at least video size parameters.

18. A computer program product in accordance with claim **11**, wherein the method further comprises:

prior to the act of transcoding, an act of determining that a transcoded representation of the requested media content transcoded using the identified transcoding parameters does not already exist.

19. A computer program product in accordance with claim **11**, wherein the act of identifying transcoding parameters comprises an act of selecting a transcoding profile from a set of pre-existing transcoding profiles.

20. A computer program product in accordance with claim **11**, wherein the act of identifying transcoding parameters comprises:

an act of reading one or more transcoding parameters from the request; and

an act of identifying the one or more transcoding parameters from the request as being the identified transcoding parameters to be applied.

* * * * *

Exhibit E

US008656046B2

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

(10) **Patent No.: US 8,656,046 B2**
(45) **Date of Patent: Feb. 18, 2014**

(54) **AUTOMATED MEDIA DELIVERY SYSTEM**

(75) Inventors: **Sean Barger**, Mill Valley, CA (US); **Steve Johnson**, Mill Valley, CA (US); **Matt Butler**, Beaverton, OR (US); **Jerry Destremps**, Sausalito, CA (US); **David Pochron**, Cambridge, WI (US); **Trent Brown**, San Anselmo, CA (US)

(73) Assignee: **Equilibrium**, Sausalito, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 384 days.

(21) Appl. No.: **12/173,747**

(22) Filed: **Jul. 15, 2008**

(65) **Prior Publication Data**

US 2009/0070485 A1 Mar. 12, 2009

Related U.S. Application Data

(60) Division of application No. 11/269,916, filed on Nov. 7, 2005, now abandoned, which is a continuation-in-part of application No. 09/929,904, filed on Aug. 14, 2001, now Pat. No. 6,964,009, which is a continuation of application No. 09/425,326, filed on Oct. 21, 1999, now Pat. No. 6,792,575.

(51) **Int. Cl.**
G06F 15/16 (2006.01)

(52) **U.S. Cl.**
USPC **709/236**; 709/231; 370/465; 370/470

(58) **Field of Classification Search**
USPC 709/227, 226, 223, 219, 206, 204, 203, 709/247, 217, 236, 246; 370/480, 467, 465, 370/389, 356, 353, 352, 289, 286, 260, 370/252; 455/560, 463, 426.1, 422.1, 455/412.1, 412; 704/503, 500, 230, 204; 382/232; 386/278, 244, 241
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,088,052	A	2/1992	Spielman et al.
5,355,472	A	10/1994	Lewis
5,442,771	A	8/1995	Filepp et al.
5,530,852	A	6/1996	Meske, Jr. et al.
5,701,451	A	12/1997	Rogers et al.
5,708,845	A	1/1998	Wistendahl et al.
5,710,918	A	1/1998	Lagarde et al.

(Continued)

FOREIGN PATENT DOCUMENTS

AU	A-53031/98	8/1996
EP	0747842	12/1996

(Continued)

OTHER PUBLICATIONS

Sakaguchi, et al.; "A browsing tool for multi-lingual documents for users without multi-lingual fonts"; 1996; ACM International Conference on Digital Libraries, pp. 63-71.

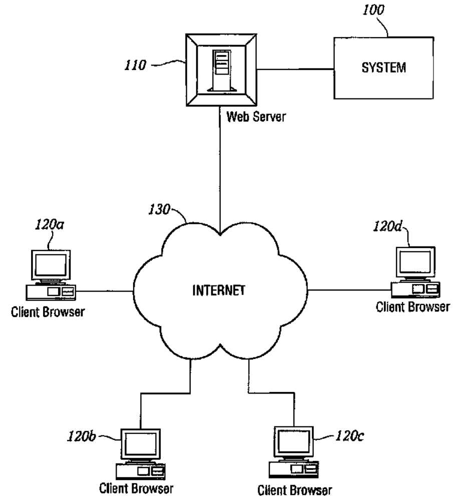
(Continued)

Primary Examiner — David Lazaro
Assistant Examiner — Charles Murphy
(74) *Attorney, Agent, or Firm* — Michael A. Glenn; Perkins Coie LLP

(57) **ABSTRACT**

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed by an author within URLs embedded within Web documents.

17 Claims, 23 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

5,737,619	A	4/1998	Judson	
5,745,908	A	4/1998	Anderson et al.	
5,758,110	A	5/1998	Boss et al.	
5,761,655	A	6/1998	Hoffman	
5,793,964	A	8/1998	Rogers et al.	
5,819,261	A	10/1998	Takahashi et al.	
5,822,436	A	10/1998	Rhoads	
5,845,084	A	12/1998	Cordell et al.	
5,845,279	A	12/1998	Garofalakis et al.	
5,845,299	A	12/1998	Arora et al.	
5,860,068	A	1/1999	Cook	
5,860,073	A	1/1999	Ferrel et al.	
5,861,881	A	1/1999	Freeman et al.	
5,862,325	A	1/1999	Reed et al.	
5,864,337	A	1/1999	Marvin	
5,870,552	A	2/1999	Dozier et al.	
5,880,740	A	3/1999	Halliday et al.	
5,890,170	A	3/1999	Sidana	
5,895,476	A	4/1999	Orr et al.	
5,895,477	A	4/1999	Orr et al.	
5,903,892	A	5/1999	Hoffert et al.	
5,937,160	A	8/1999	Davis et al.	
5,943,680	A	8/1999	Ohga et al.	
5,956,737	A	9/1999	King et al.	
6,009,436	A	12/1999	Motoyama et al.	
6,456,305	B1	9/2002	Qureshi et al.	
6,463,445	B1 *	10/2002	Suzuki et al.	1/1
6,483,851	B1 *	11/2002	Neogi	370/466
6,484,149	B1	11/2002	Jemmes et al.	
6,563,517	B1 *	5/2003	Bhagwat et al.	715/735
6,591,280	B2	7/2003	Orr	
6,623,529	B1	9/2003	Lakritz	
6,909,708	B1 *	6/2005	Krishnaswamy et al.	370/352
6,938,073	B1	8/2005	Mendhekar et al.	
7,284,201	B2	10/2007	Cohen-Solal	
7,313,361	B2	12/2007	Steelberg et al.	
7,406,434	B1	7/2008	Chang et al.	
7,477,688	B1	1/2009	Zhang et al.	
7,673,063	B2	3/2010	Xie et al.	
2003/0225568	A1	12/2003	Salmonsen	
2004/0025176	A1	2/2004	Franklin et al.	
2005/0091311	A1 *	4/2005	Lund et al.	709/203
2005/0255852	A1	11/2005	Steelberg et al.	
2005/0278794	A1	12/2005	Leinonen et al.	
2006/0015580	A1	1/2006	Gabriel et al.	
2006/0127059	A1	6/2006	Fanning	
2007/0061198	A1	3/2007	Ramer et al.	
2007/0234213	A1	10/2007	Krikorian et al.	
2008/0155230	A1	6/2008	Robbins et al.	
2008/0186377	A1	8/2008	Eriksson et al.	
2008/0195938	A1	8/2008	Tischer et al.	
2008/0205389	A1	8/2008	Fang et al.	
2008/0207182	A1	8/2008	Maharajh et al.	
2009/0013347	A1	1/2009	Ahanger et al.	
2009/0089422	A1	4/2009	Barger et al.	
2009/0240569	A1	9/2009	Ramer et al.	
2009/0254672	A1	10/2009	Zhang	
2010/0046842	A1	2/2010	Conwell	
2010/0153495	A1	6/2010	Barger et al.	
2011/0221745	A1	9/2011	Goldman et al.	
2011/0279638	A1	11/2011	Periyannan et al.	
2012/0016858	A1	1/2012	Rathod	

FOREIGN PATENT DOCUMENTS

EP	0782085	7/1997
EP	0818907	1/1998
EP	0843276	5/1998
EP	0876034	11/1998
EP	0883068	12/1998
EP	0886409	12/1998
EP	0895171	2/1999
EP	0926607	6/1999
EP	0949571	10/1999
WO	WO 97/49252	12/1997
WO	WO 98/40842	9/1998
WO	WO 98/43177	10/1998

OTHER PUBLICATIONS

Zaiane, et al.; “Mining multimedia data”; Nov. 1998; ACM Conference of the Center for Advanced Studies on Collaborative research, pp. 1-18.

Bulterman, Dick.C.A.; *Models, Media and Motion: Using the Web to Support Multimedia Documents*; Proceedings of 1997 Intnl Conf on Multimedia Modeling; p. 17-20; Nov. 1997; Singapore.

Mohler, J.L.; *Migrating Course Materials to the World Wide Web: A Case Study of the Department of Techinal Graphics at Purdue University*; Computer Networks and ISDN Systems; vol. 30, Issues 20-21, p. 1981-1990; Nov. 12, 1988.

Dobson, R.; *Animating Your Web Pages with Direct Animation*; Web Techniques; vol. 3, No. 6, p. 49-52; Jun. 1998.

Berinstein, Paula; “The Big Picture; Text and Graphics on UMI’s ProQuest Direct: The Best (Yet) of Both Words”; Mar. 1997; retrieved on Mar. 23, 2004 from website: http://www.infotoday.com/online/MarOL97/picture3.html.

McNeil, Sara; Research Interests; retrieved on Mar. 18, 2004 from website: http//www.coe.uh.edu/’smcneil/research.htm.

Tables of Contents service for Computers & Geosciences; Copyright 1997; Computers and GeoSciences, vol. 23, Issue 5, retrieved on Mar. 18, 2004 from website: http://library.iem.ac.ru/comp&geo/00983004/sz977014.html.

* cited by examiner

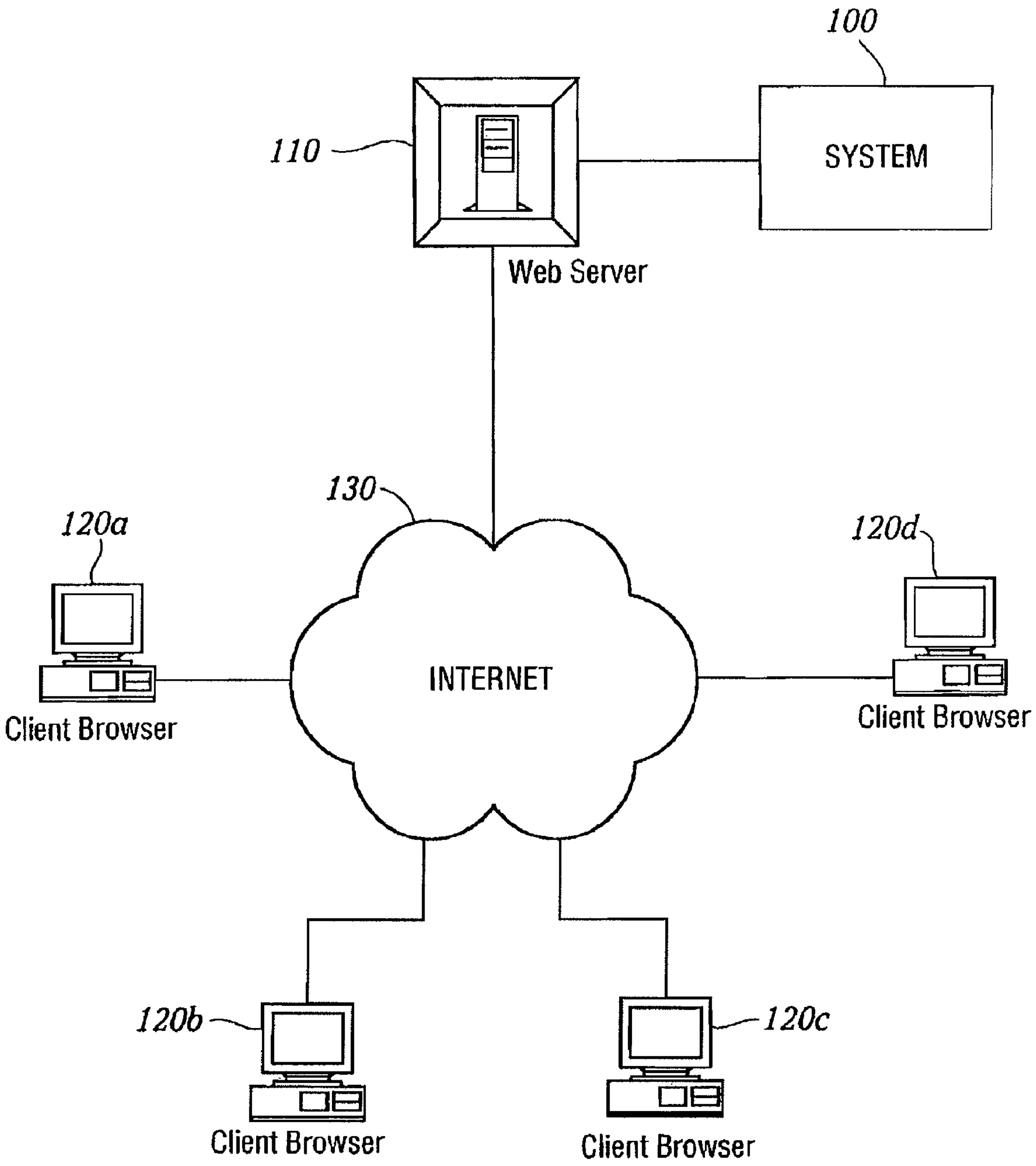

100
110
SYSTEM
Web Server
130
INTERNET
120a
Client Browser
120d
Client Browser
120b
Client Browser
120c
Client Browser

FIG. 1

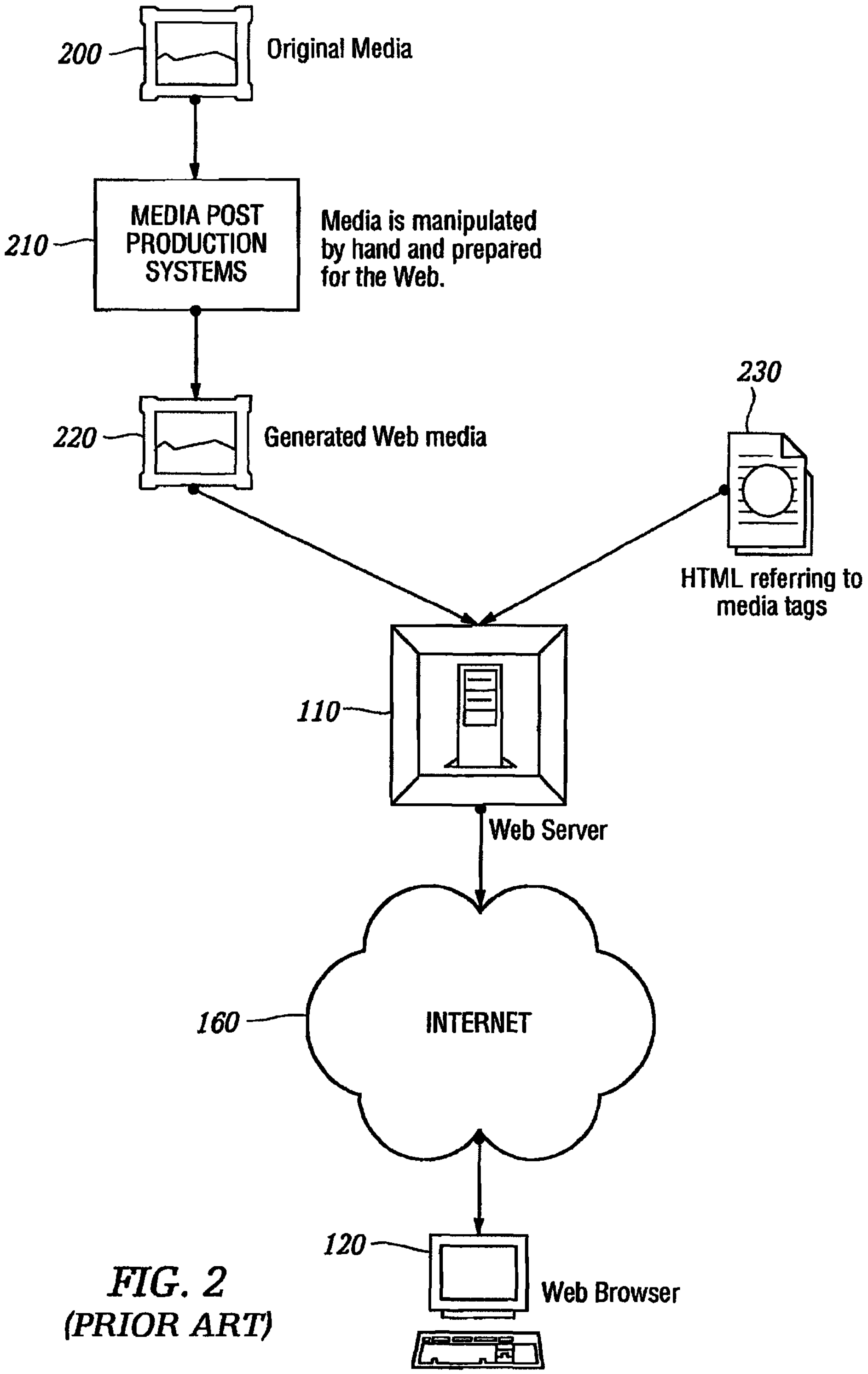

200
Original Media
210
MEDIA POST PRODUCTION SYSTEMS
Media is manipulated by hand and prepared for the Web.
220
Generated Web media
230
HTML referring to media tags
110
Web Server
160
INTERNET
120
Web Browser

FIG. 2
(PRIOR ART)

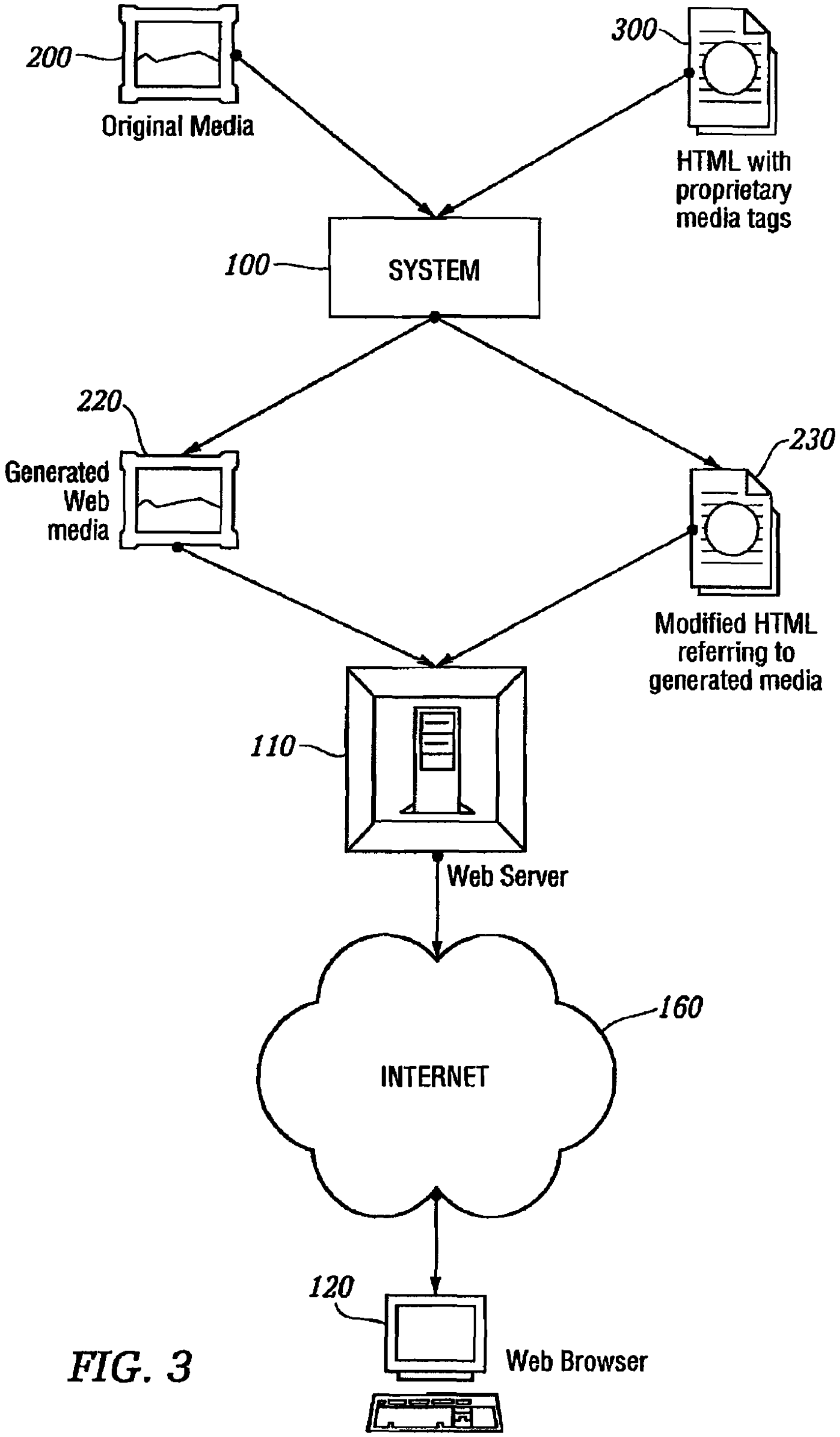

200
Original Media
300
HTML with proprietary media tags
100
SYSTEM
220
Generated Web media
230
Modified HTML referring to generated media
110
Web Server
160
INTERNET
120
Web Browser

FIG. 3

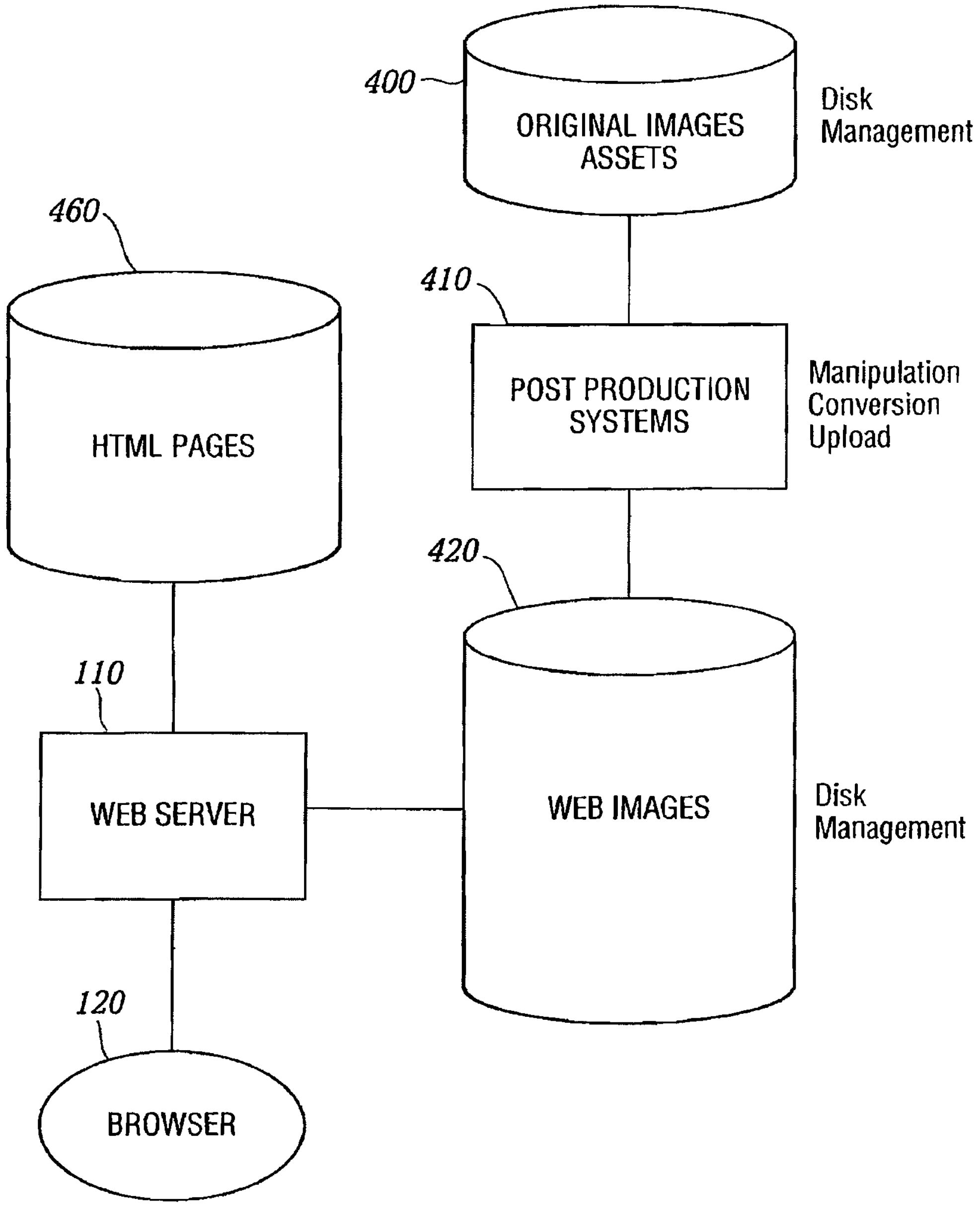


FIG. 4
(PRIOR ART)

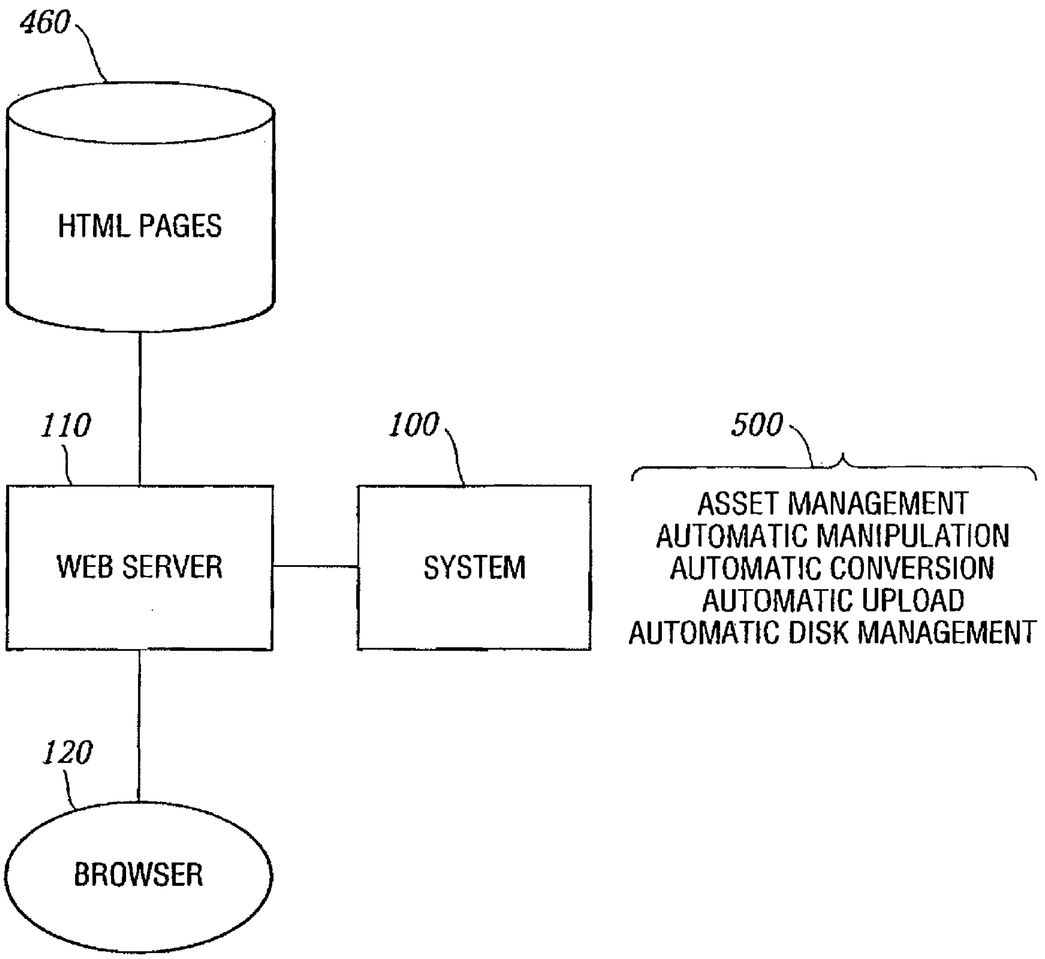

460
HTML PAGES
110
WEB SERVER
100
SYSTEM
500
ASSET MANAGEMENT
AUTOMATIC MANIPULATION
AUTOMATIC CONVERSION
AUTOMATIC UPLOAD
AUTOMATIC DISK MANAGEMENT
120
BROWSER

FIG. 5

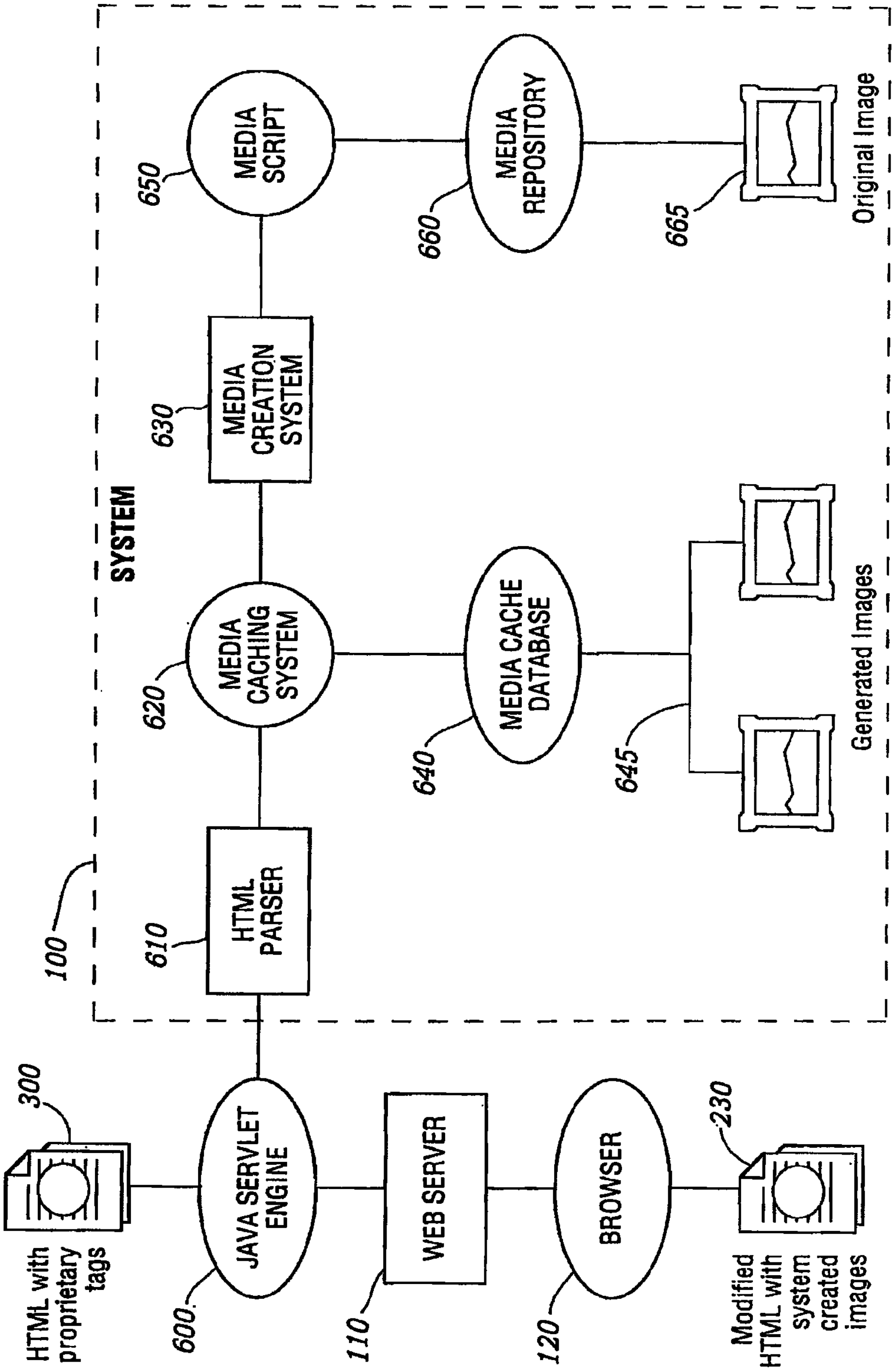

100
SYSTEM
610
HTML PARSER
620
MEDIA CACHING SYSTEM
630
MEDIA CREATION SYSTEM
650
MEDIA SCRIPT
640
MEDIA CACHE DATABASE
660
MEDIA REPOSITORY
645
Generated Images
665
Original Image
300
HTML with proprietary tags
600
JAVA SERVLET ENGINE
110
WEB SERVER
120
BROWSER
230
Modified HTML with system created images

FIG. 6

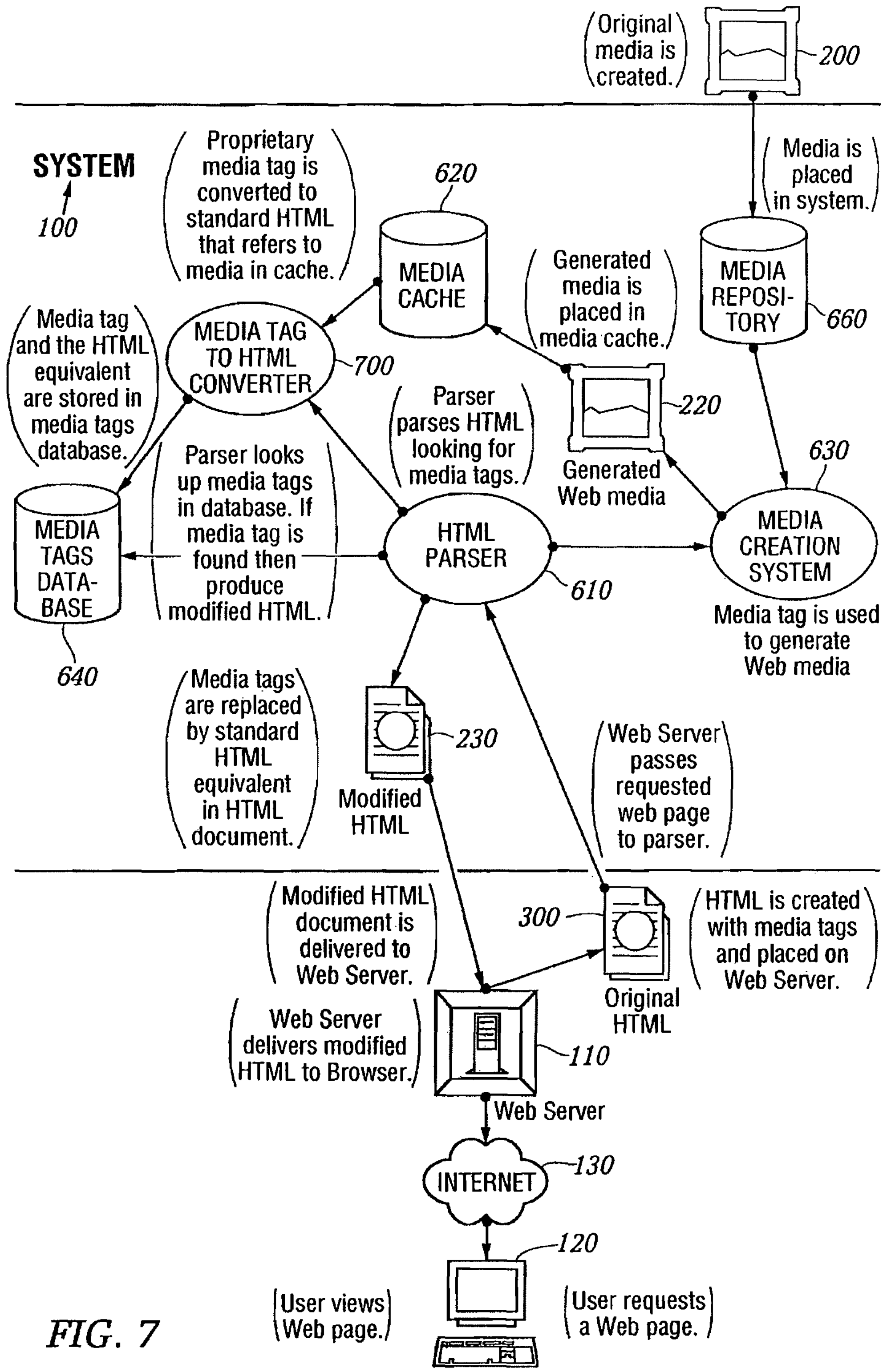

Original media is created.
200
SYSTEM
100
Proprietary media tag is converted to standard HTML that refers to media in cache.
620
MEDIA CACHE
Media is placed in system.
MEDIA REPOSI-TORY
660
Generated media is placed in media cache.
Media tag and the HTML equivalent are stored in media tags database.
MEDIA TAG TO HTML CONVERTER
700
Parser parses HTML looking for media tags.
220
Generated Web media
630
Parser looks up media tags in database. If media tag is found then produce modified HTML.
MEDIA TAGS DATA-BASE
HTML PARSER
MEDIA CREATION SYSTEM
610
Media tag is used to generate Web media
640
Media tags are replaced by standard HTML equivalent in HTML document.
230
Modified HTML
Web Server passes requested web page to parser.
Modified HTML document is delivered to Web Server.
300
HTML is created with media tags and placed on Web Server.
Original HTML
Web Server delivers modified HTML to Browser.
110
Web Server
INTERNET
130
120
User views Web page.
User requests a Web page.

FIG. 7

AUTHORING FLOWCHART

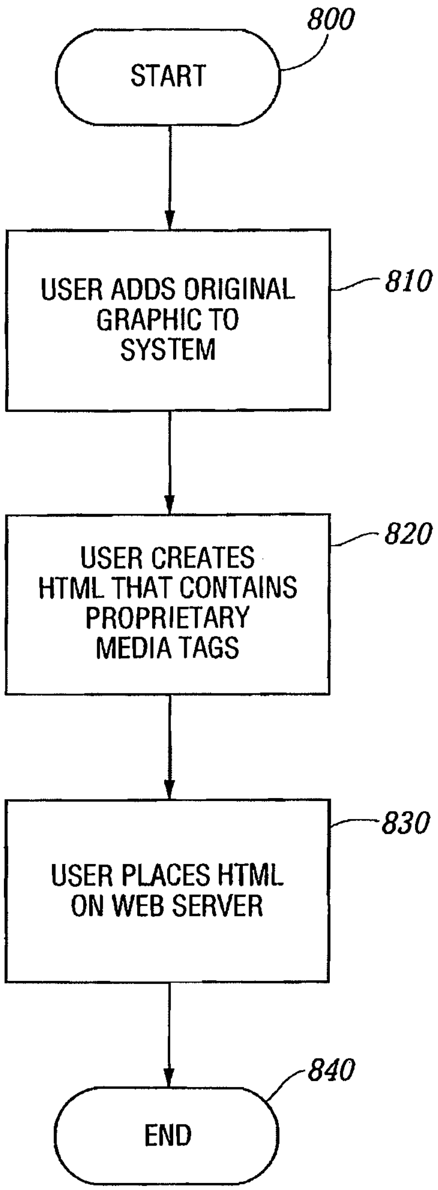

800
START
USER ADDS ORIGINAL GRAPHIC TO SYSTEM
810
USER CREATES HTML THAT CONTAINS PROPRIETARY MEDIA TAGS
820
USER PLACES HTML ON WEB SERVER
830
840
END

FIG. 8

HTML PARSING FLOWCHART

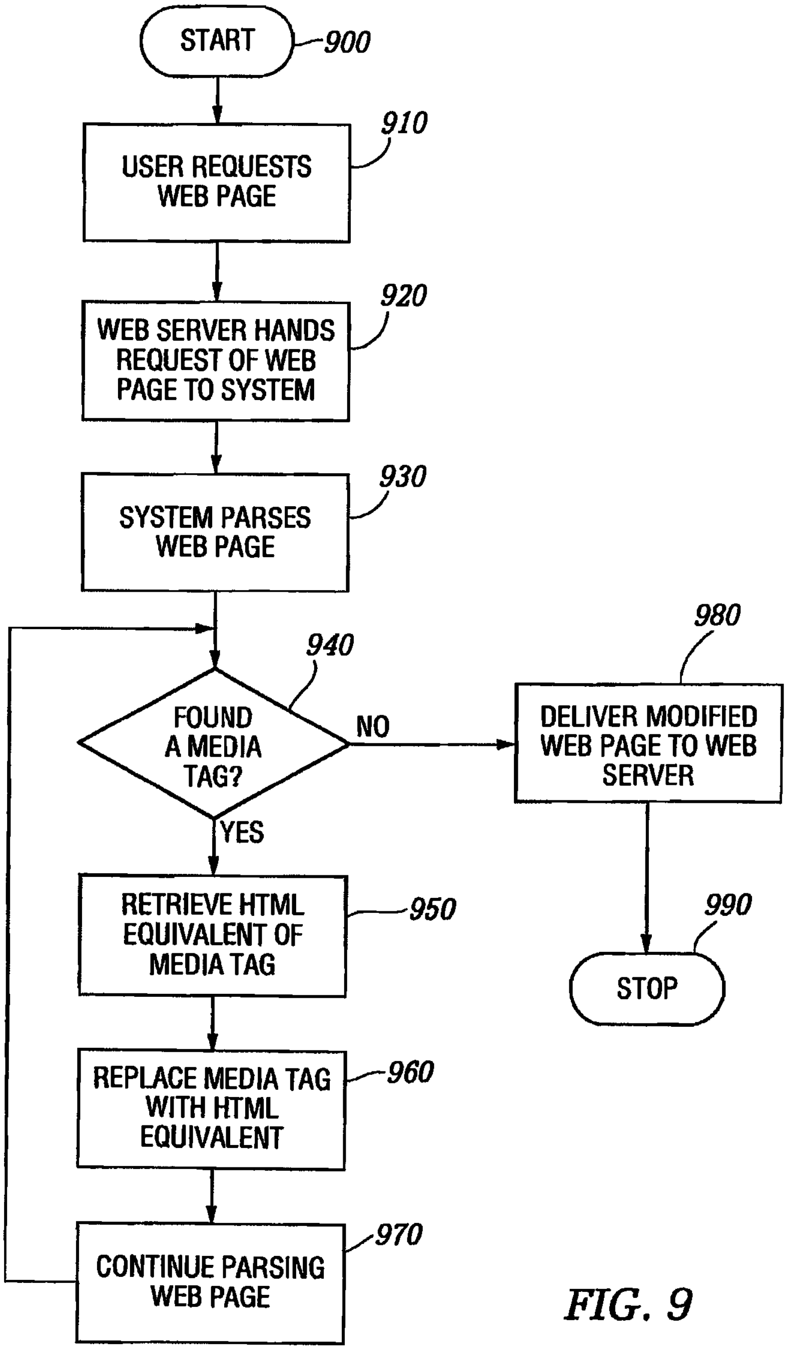

START
900
USER REQUESTS WEB PAGE
910
WEB SERVER HANDS REQUEST OF WEB PAGE TO SYSTEM
920
SYSTEM PARSES WEB PAGE
930
FOUND A MEDIA TAG?
940
NO
YES
DELIVER MODIFIED WEB PAGE TO WEB SERVER
980
STOP
990
RETRIEVE HTML EQUIVALENT OF MEDIA TAG
950
REPLACE MEDIA TAG WITH HTML EQUIVALENT
960
CONTINUE PARSING WEB PAGE
970

FIG. 9

MEDIA CREATION FLOWCHART

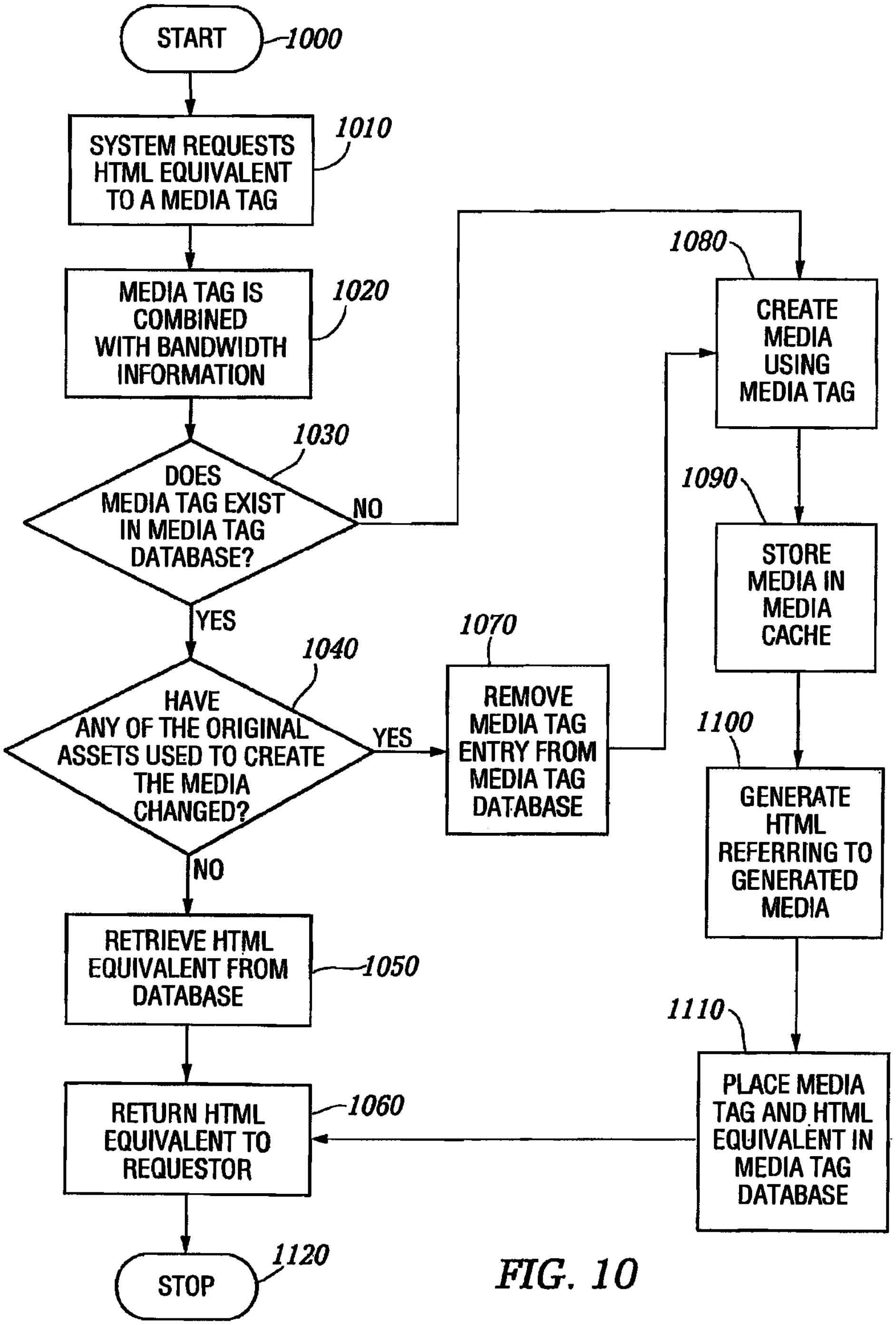
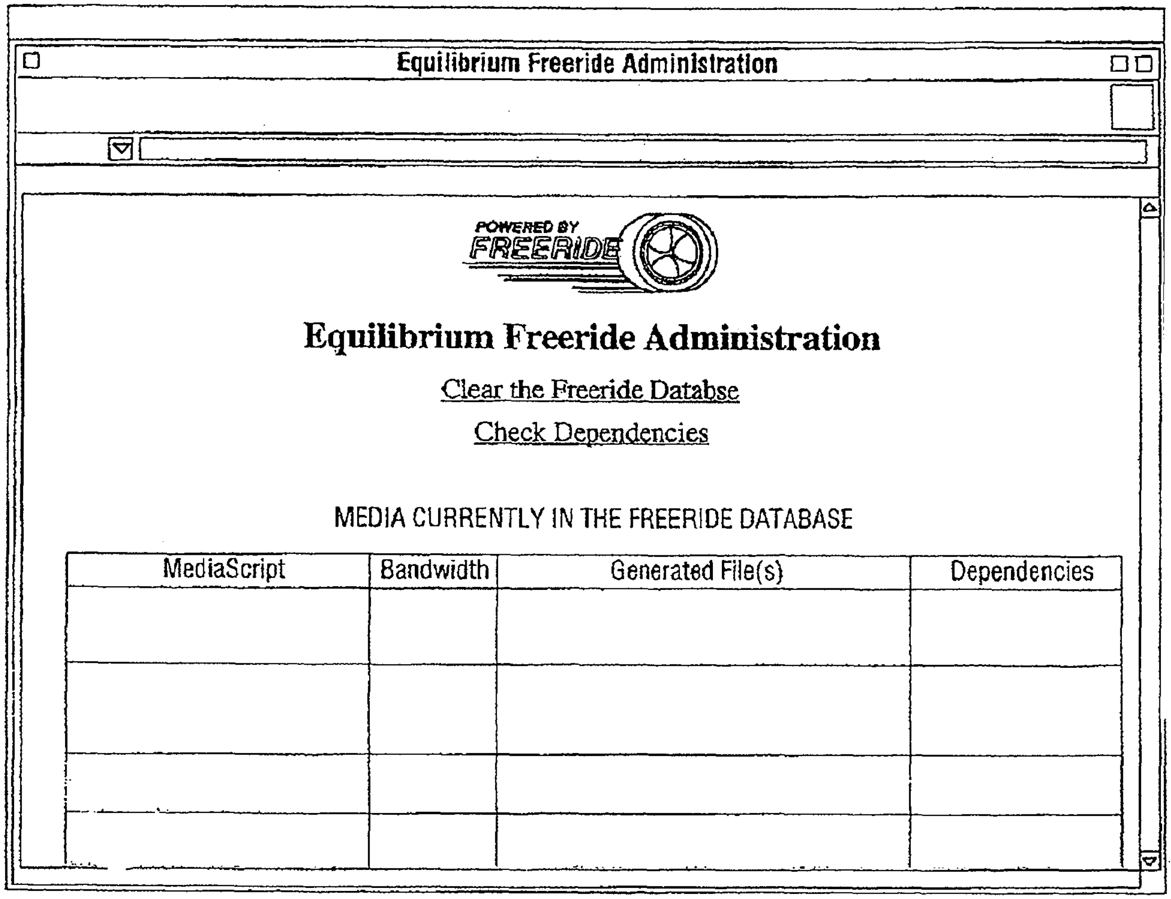

START
1000
SYSTEM REQUESTS HTML EQUIVALENT TO A MEDIA TAG
1010
MEDIA TAG IS COMBINED WITH BANDWIDTH INFORMATION
1020
DOES MEDIA TAG EXIST IN MEDIA TAG DATABASE?
1030
NO
YES
HAVE ANY OF THE ORIGINAL ASSETS USED TO CREATE THE MEDIA CHANGED?
1040
YES
NO
RETRIEVE HTML EQUIVALENT FROM DATABASE
1050
RETURN HTML EQUIVALENT TO REQUESTOR
1060
STOP
1120
REMOVE MEDIA TAG ENTRY FROM MEDIA TAG DATABASE
1070
CREATE MEDIA USING MEDIA TAG
1080
STORE MEDIA IN MEDIA CACHE
1090
GENERATE HTML REFERRING TO GENERATED MEDIA
1100
PLACE MEDIA TAG AND HTML EQUIVALENT IN MEDIA TAG DATABASE
1110

FIG. 10



Equilibrium Freeride Administration

Equilibrium Freeride Administration

Clear the Freeride Databse

Check Dependencies

MEDIA CURRENTLY IN THE FREERIDE DATABASE

MediaScript	Bandwidth	Generated File(s)	Dependencies

FIG. 11

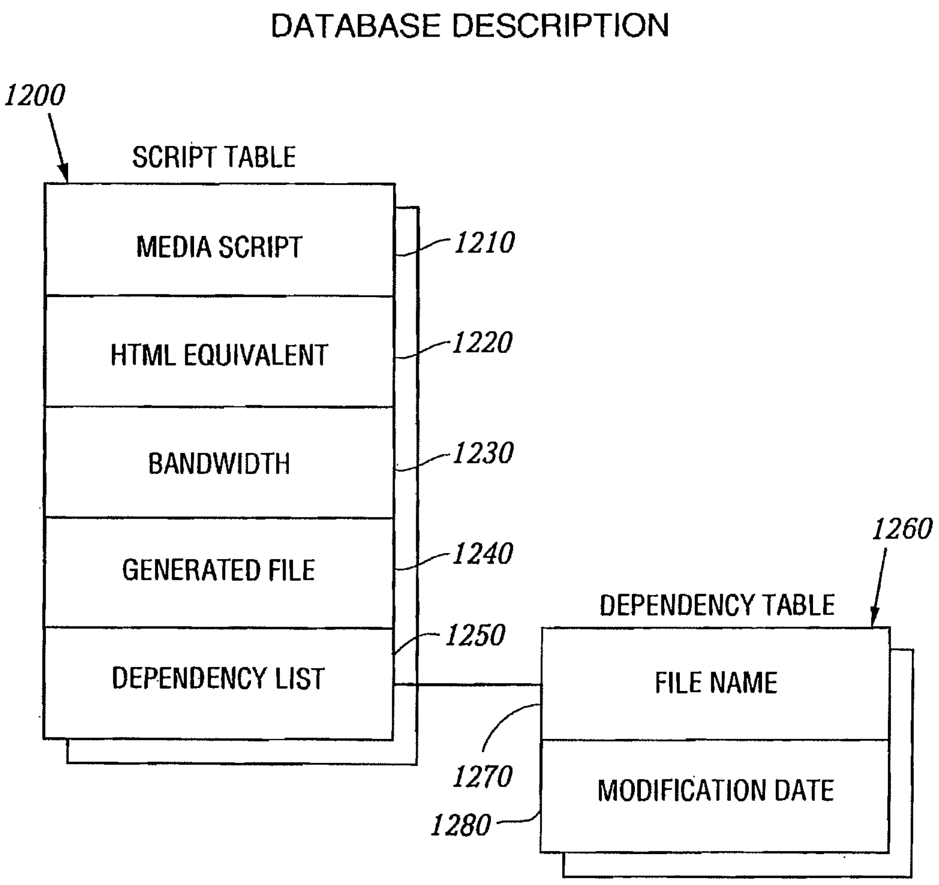

DATABASE DESCRIPTION
1200
SCRIPT TABLE
MEDIA SCRIPT
1210
HTML EQUIVALENT
1220
BANDWIDTH
1230
GENERATED FILE
1240
DEPENDENCY LIST
1250
1260
DEPENDENCY TABLE
FILE NAME
1270
MODIFICATION DATE
1280

FIG. 12

ORIGINAL IMAGES

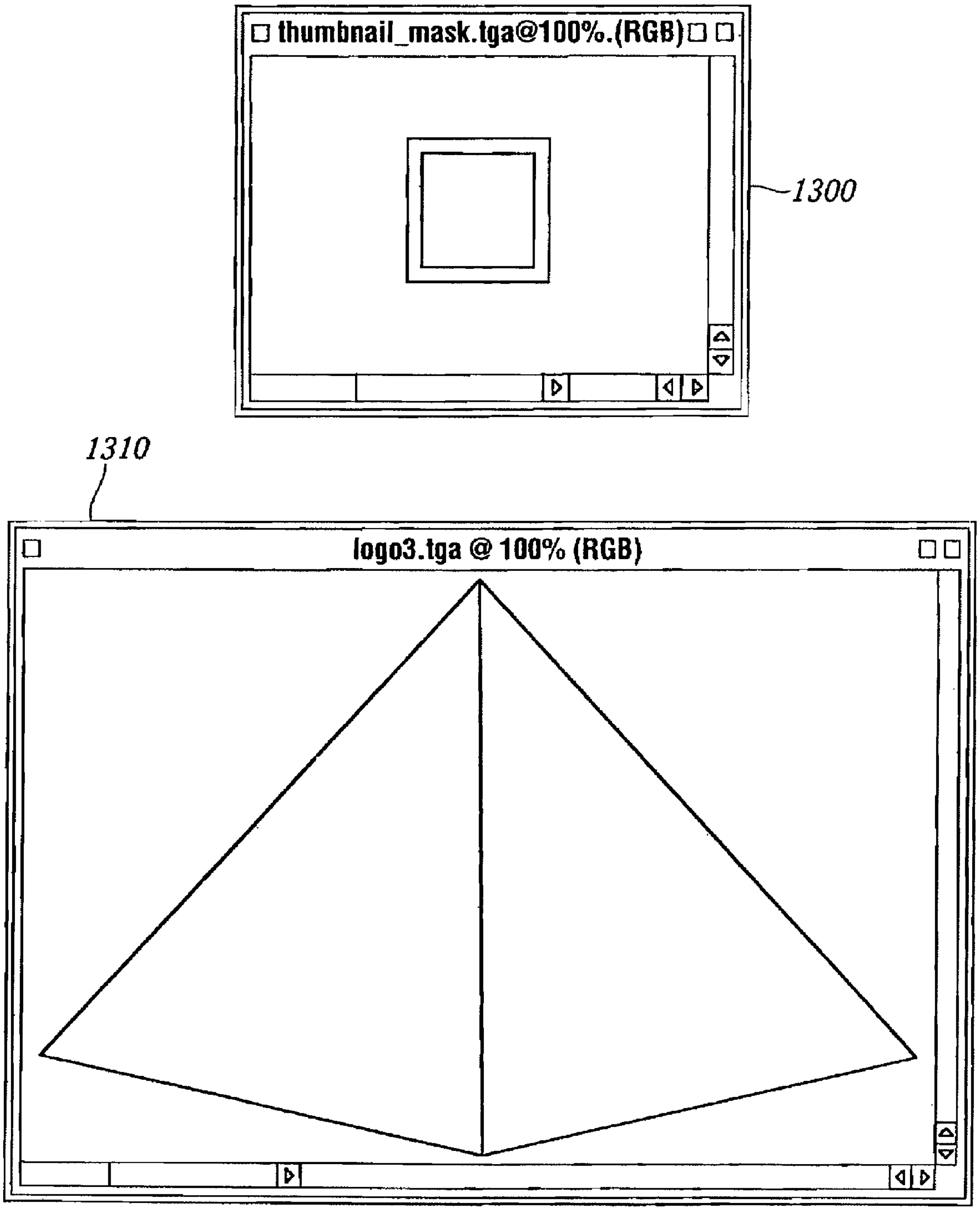

□ thumbnail_mask.tga@100%.(RGB) □ □
1300
1310
□ logo3.tga @ 100% (RGB) □ □

FIG.13

HTML DOCUMENT WITH PROPRIETARY TAG

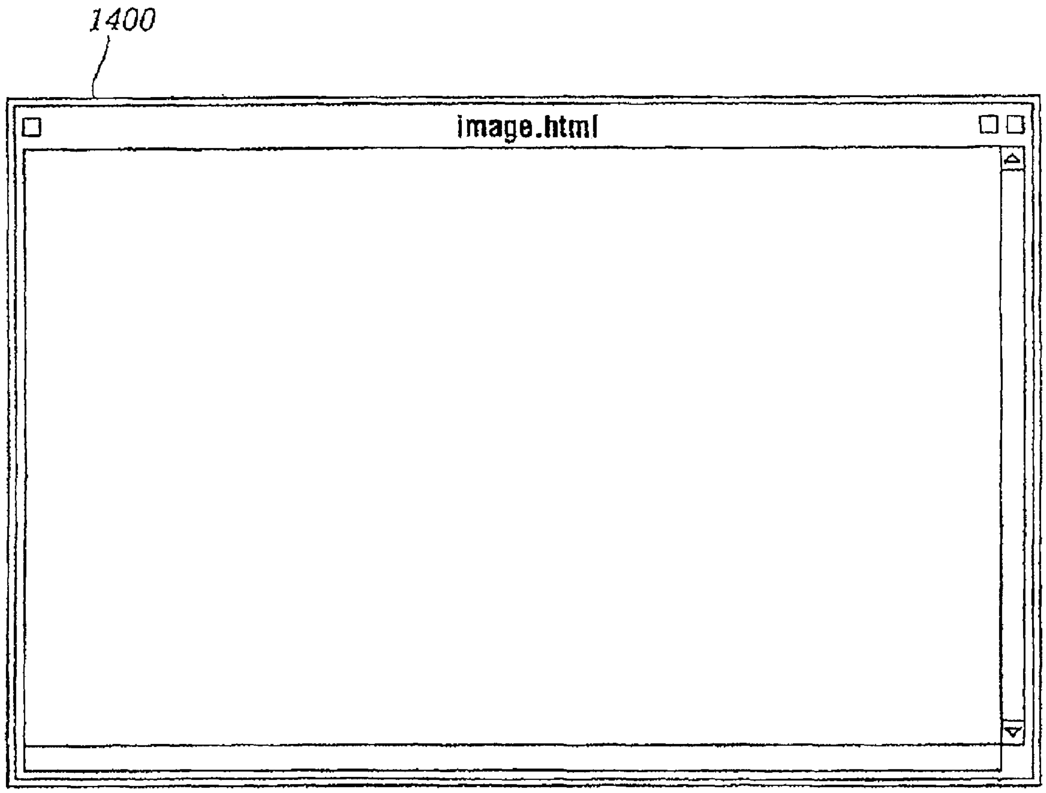


FIG.14

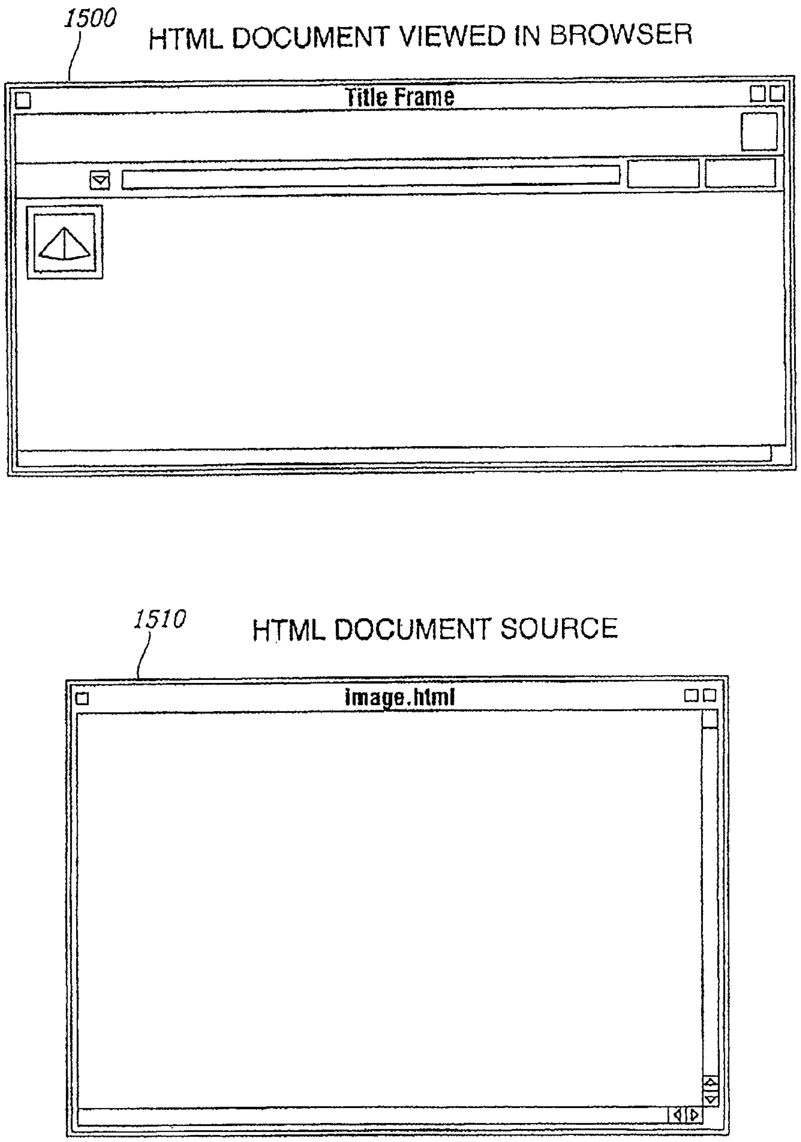

1500
HTML DOCUMENT VIEWED IN BROWSER
Title Frame
1510
HTML DOCUMENT SOURCE
image.html

FIG.15

GENERATED GIF IMAGE

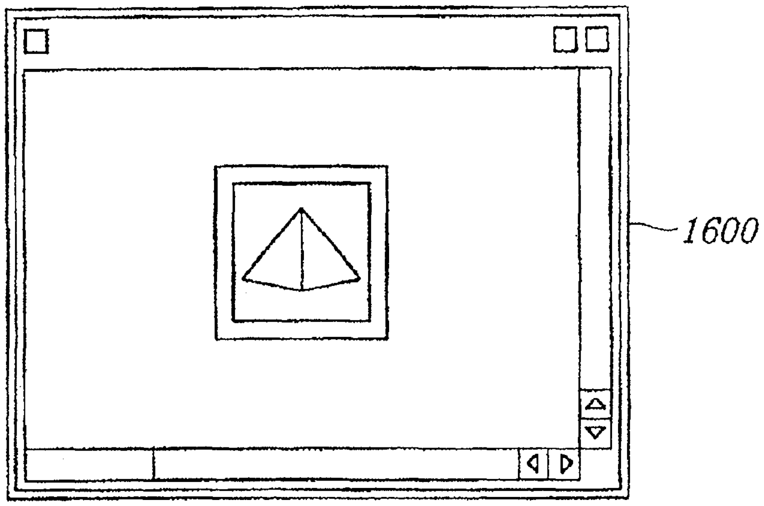


FIG.16

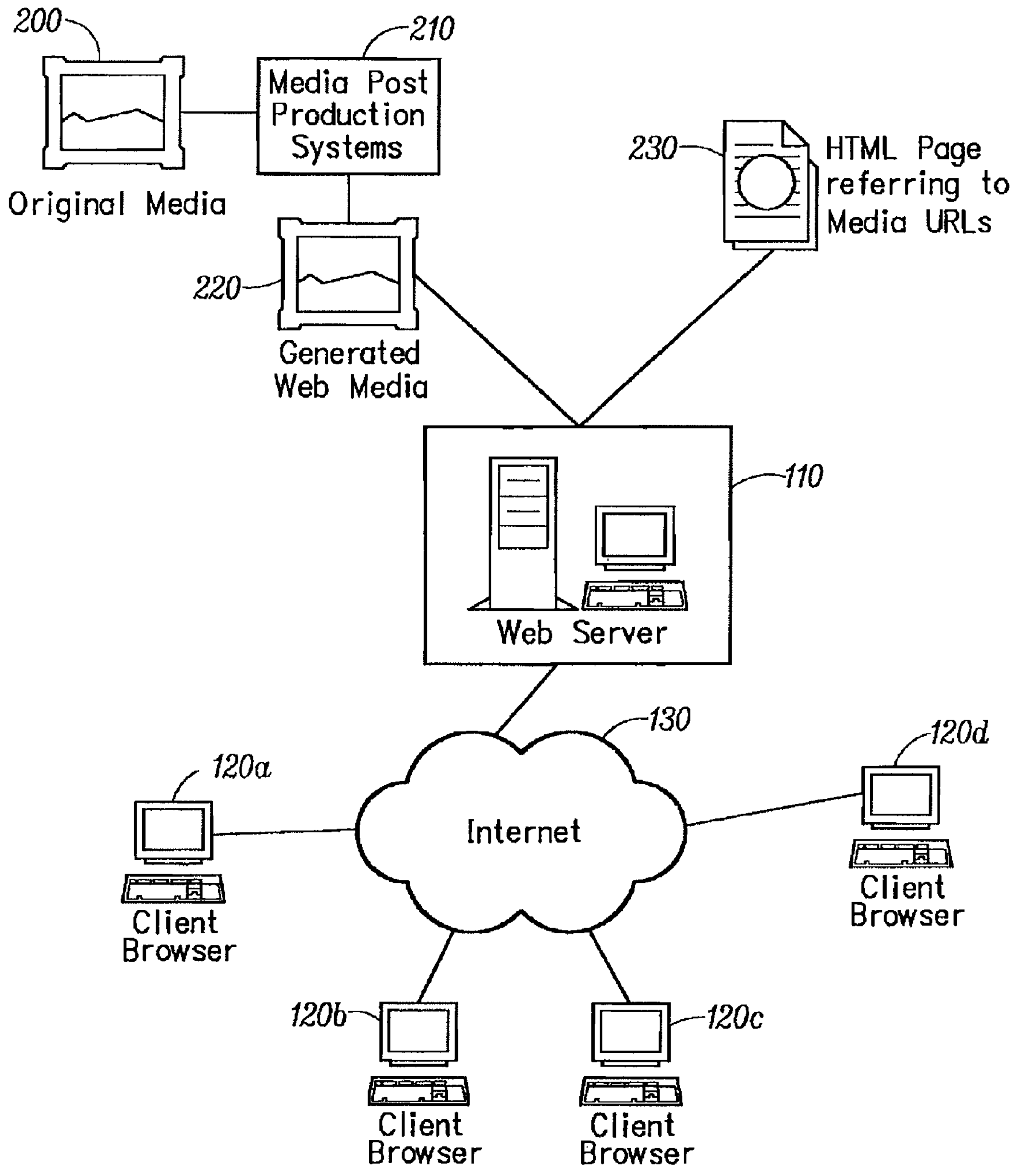

200
Original Media
210
Media Post Production Systems
230
HTML Page referring to Media URLs
220
Generated Web Media
110
Web Server
130
Internet
120a
Client Browser
120d
Client Browser
120b
Client Browser
120c
Client Browser

FIG. 17

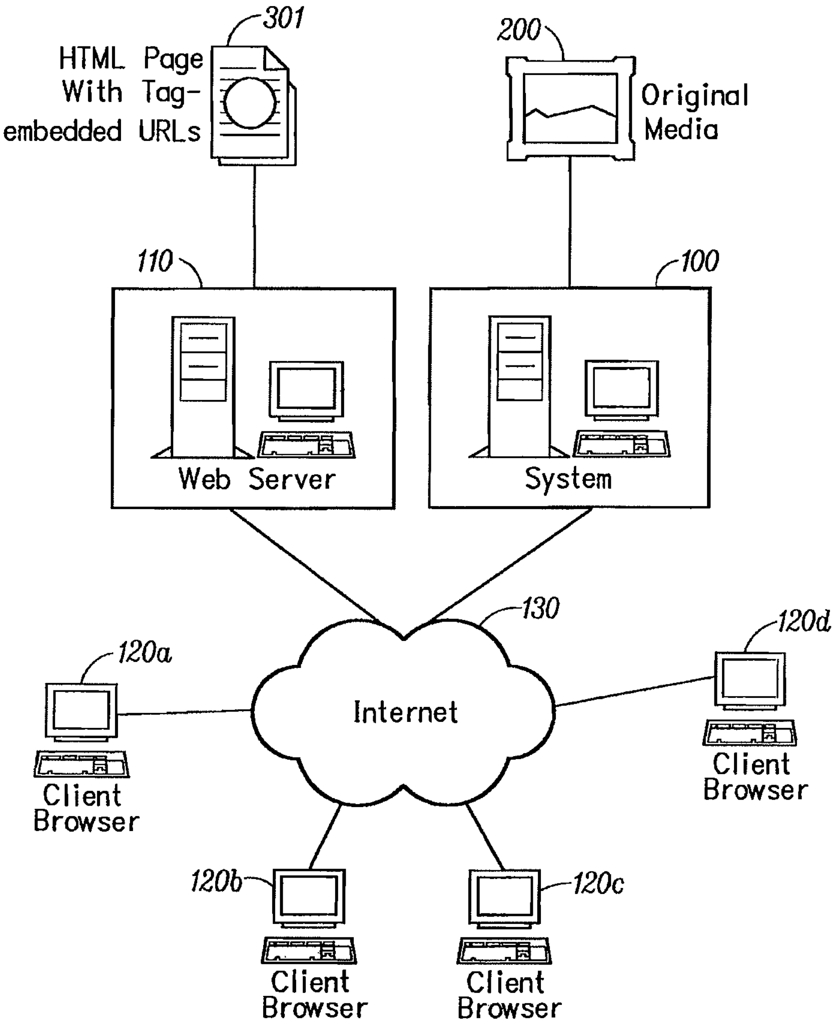

301
HTML Page
With Tag-
embedded URLs
200
Original
Media
110
100
Web Server
System
130
Internet
120a
Client
Browser
120d
Client
Browser
120b
Client
Browser
120c
Client
Browser

FIG. 18

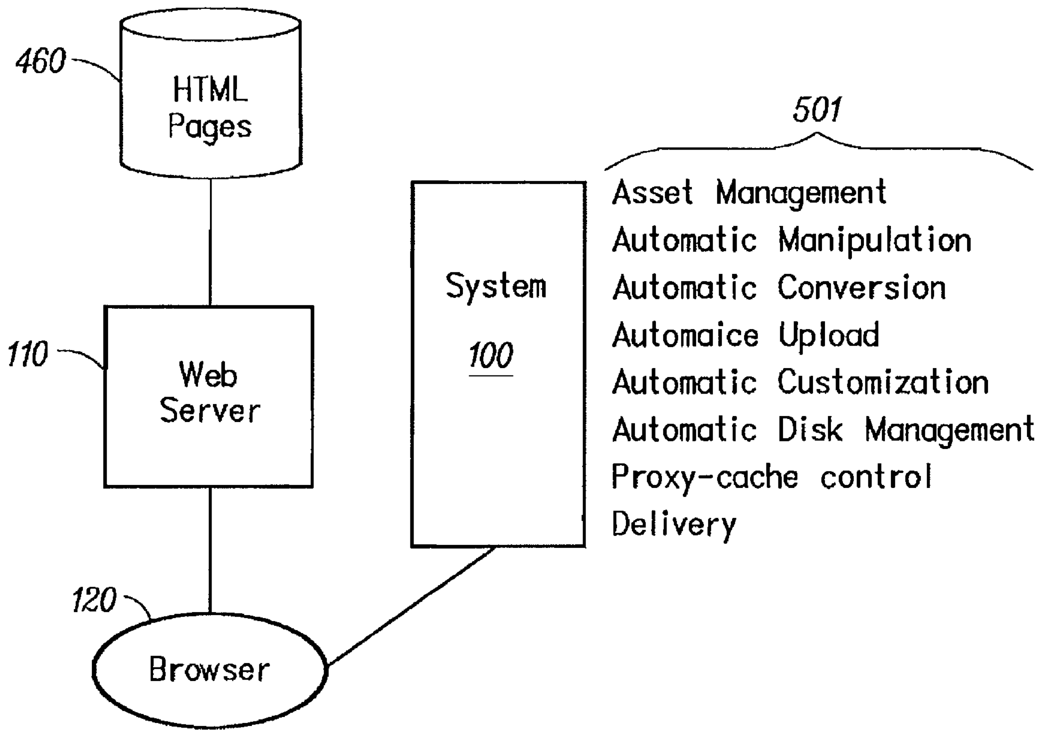

460
HTML Pages
110
Web Server
120
Browser
System
100
501
Asset Management
Automatic Manipulation
Automatic Conversion
Automaice Upload
Automatic Customization
Automatic Disk Management
Proxy-cache control
Delivery

FIG. 19

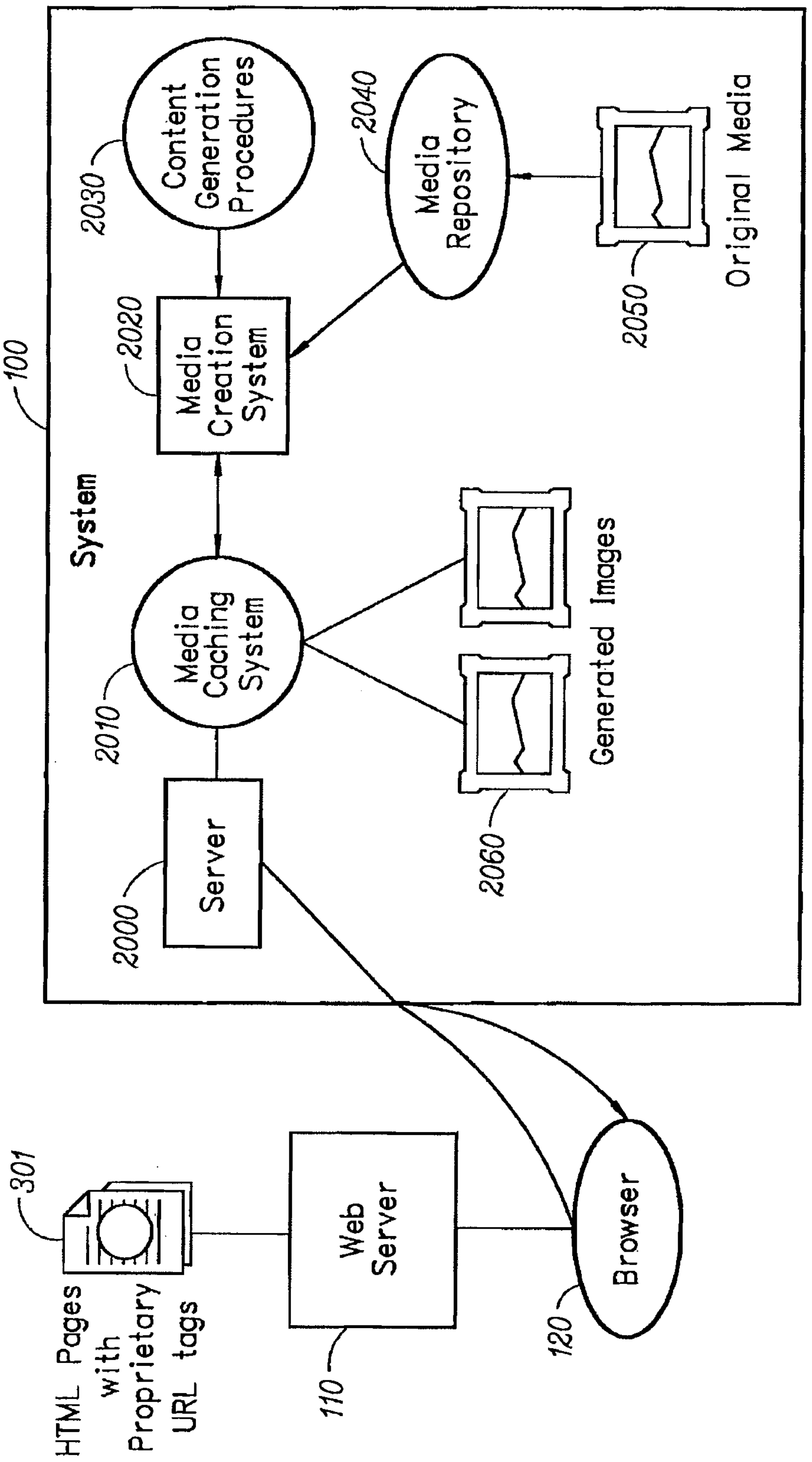

100
System
Content Generation Procedures
2030
Media Repository
2040
Media Creation System
2020
Original Media
2050
Media Caching System
2010
Generated Images
2060
Server
2000
HTML Pages with Proprietary URL tags
301
Web Server
110
Browser
120

FIG. 20

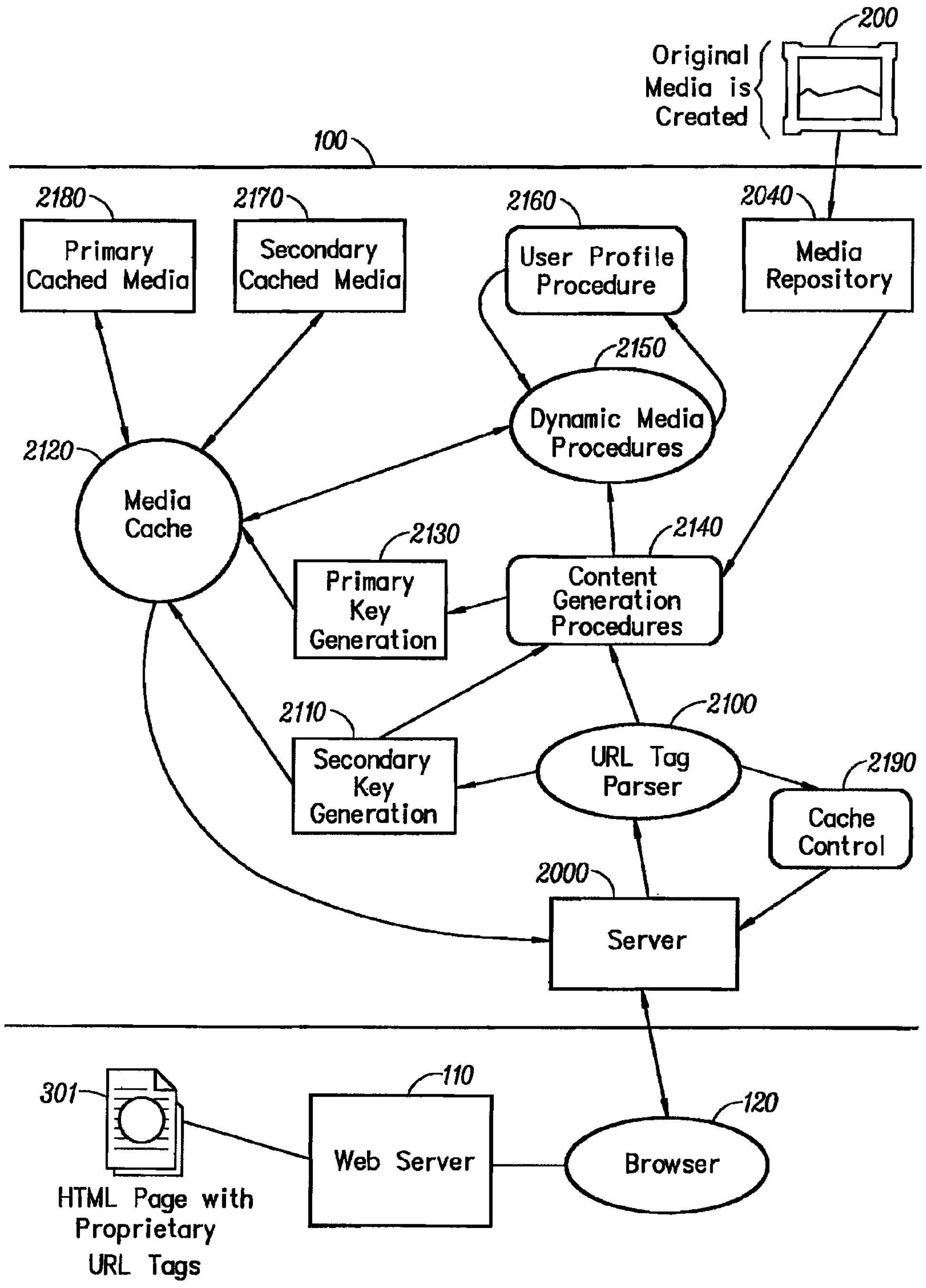

200
Original Media is Created
100
2180
Primary Cached Media
2170
Secondary Cached Media
2160
User Profile Procedure
2040
Media Repository
2150
Dynamic Media Procedures
2120
Media Cache
2130
Primary Key Generation
2140
Content Generation Procedures
2110
Secondary Key Generation
2100
URL Tag Parser
2190
Cache Control
2000
Server
301
HTML Page with Proprietary URL Tags
110
Web Server
120
Browser

FIG. 21

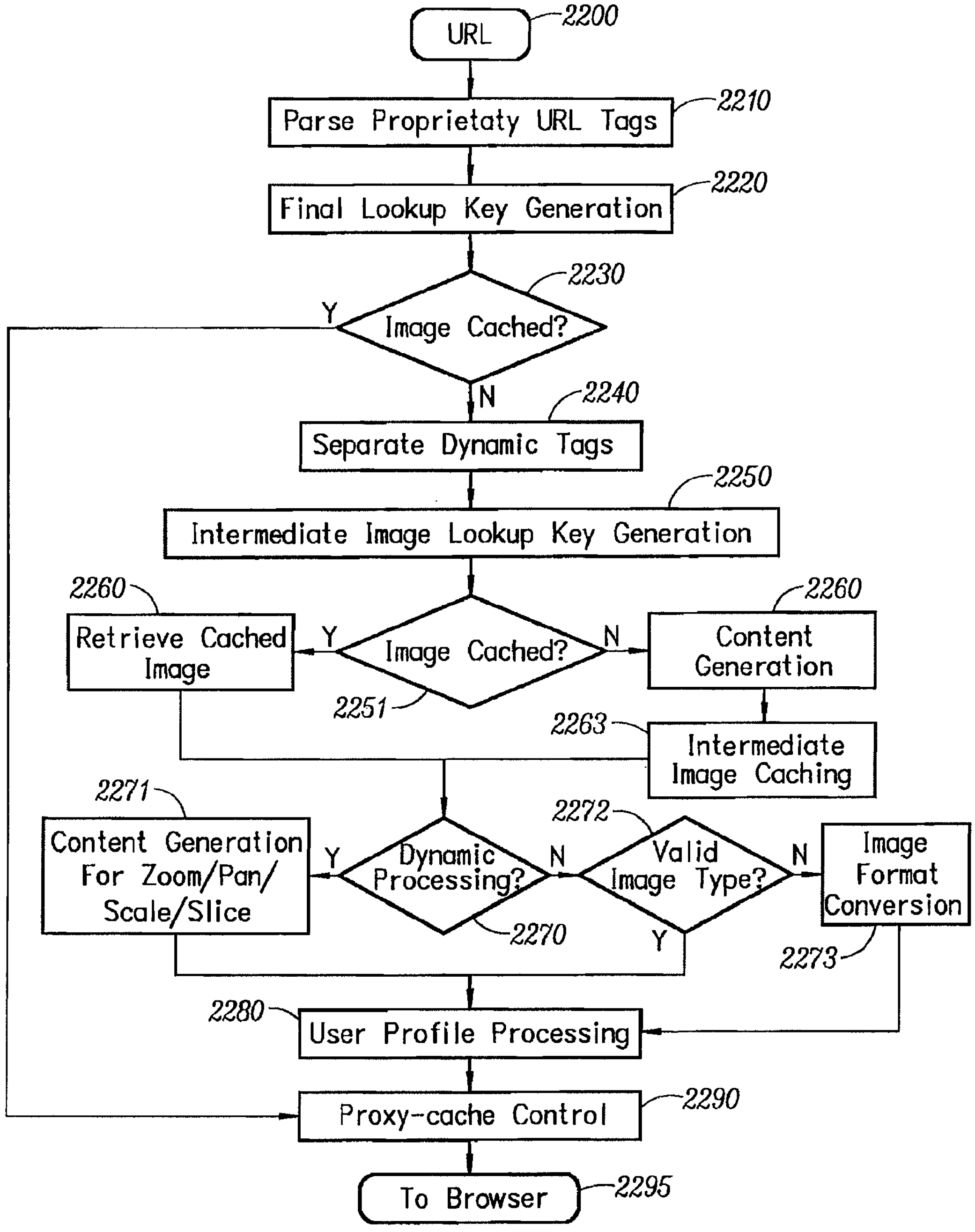

URL
2200
Parse Proprietaty URL Tags
2210
Final Lookup Key Generation
2220
Image Cached?
2230
Y
N
Separate Dynamic Tags
2240
Intermediate Image Lookup Key Generation
2250
2260
Retrieve Cached Image
Y
Image Cached?
N
2260
Content Generation
2251
2263
Intermediate Image Caching
2271
Content Generation For Zoom/Pan/ Scale/Slice
Y
Dynamic Processing?
N
2272
Valid Image Type?
N
Image Format Conversion
2270
Y
2273
2280
User Profile Processing
Proxy-cache Control
2290
To Browser
2295

FIG. 22

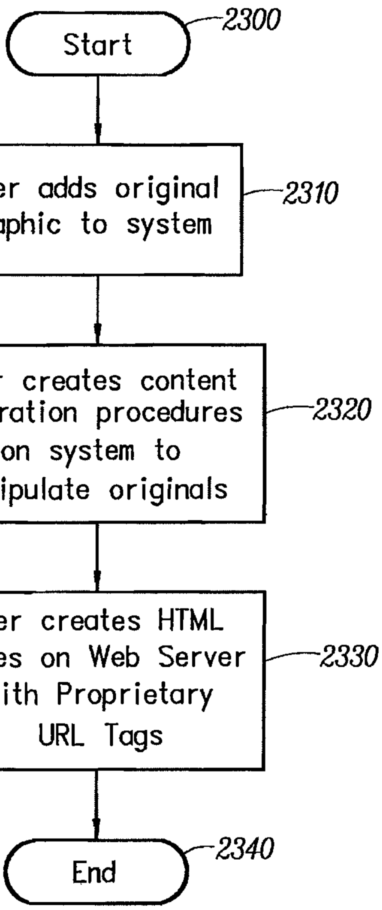

Start
2300
User adds original graphic to system
2310
User creates content generation procedures on system to manipulate originals
2320
User creates HTML pages on Web Server with Proprietary URL Tags
2330
End
2340

FIG. 23

AUTOMATED MEDIA DELIVERY SYSTEM

CROSS REFERENCE TO RELATED APPLICATIONS

This application is a Divisional of U.S. Ser. No. 11/269,916, filed Nov. 7, 2005 now abandoned, which is a Continuation-in-Part of U.S. Ser. No. 09/929,904, filed Aug. 14, 2001, now U.S. Pat. No. 6,964,009 granted on Nov. 8, 2005, which is a Continuation of U.S. Ser. No. 09/425,326, filed Oct. 21, 1999, now U.S. Pat. No. 6,792,575, granted on Sep. 14, 2004, all of which are hereby incorporated in its entirety by reference.

BACKGROUND OF THE INVENTION

1. Technical Field

The invention relates to software systems. More particularly, the invention relates to an Internet server-based software system that provides delivery of automated graphics and other media to Web sites for access by an end user or consumer.

2. Description of the Prior Art

Most Web sites today are primarily handmade. From the guy publishing a simple online technology newsletter from his home, to the Fortune 1000 company's multi-tiered site with hundreds of pages of text, images, and animations, the Web developer and each of his HTML-coding and graphics-producing coworkers toil page by page and image by image. Thousands of established online companies employ hundreds of highly-skilled workers just to produce and maintain their Web sites. After all, the Web is now a major selling vehicle and marketing medium for many of these companies. The Web has even sprouted service industries such as, for example, public companies with multi-billion dollar valuations created just to consult and produce Web sites for others.

Most Web developers who use established WYSIWYG tools in the industry still must produce each page on their Web site one by one. The same rate applies to preparing and placing images, animations, and other visual assets. Each page represents its own set of issues ranging from whether to use GIF, JPEG, or PNG file formats, to finding the optimum bit depth for each image to ensure the fastest downloading through the different browsers of the consumer. The bottlenecked state of the customer's workflow to produce graphics for Web pages can be described as follows:

Current Workflow for Creating Web Graphics
Original Artwork/Asset Creation
  Use third-party point products
Asset Editing
  Scale/reduce/slice
Asset Format Conversion
  JPEG/GIF/PNG
Asset Staging
  Place in Web file system
  Edit HTML
Create/Modify HTML for particular page
Store HTML on Web server
View final pages
Repeat process for each version of each graphic on each page
Estimated Time
Two hours per page times the number of pages

Also, from a user's perspective, the current state of the art is to offer the consumer zooming and panning capabilities so that by clicking on an image the consumer can view more closely or from a different angle. On the horizon are pages

with three-dimensional imagery that enable a user to move around a page that can look more like a room than a brochure. While interesting, these features are merely incremental improvements to a consumer's surfing experience.

D. C. A. Bulterman, *Models, Media, and Motion: Using the Web to Support Multimedia Documents*, Proceedings of 1997 International Conference on Multimedia Modeling, Singapore, 17-20 Nov. 1997 discloses "an effort underway by members of industry, research centers and user groups to define a standard document format that can be used in conjunction with time-based transport protocols over the Internet and intranets to support rich multimedia presentations. The paper outlines the goals of the W3C's Synchronized Multimedia working group and presents an initial description of the first version of the proposed multimedia document model and format."

Text and Graphics on UMI's ProQuest Direct: The Best (yet) of both Worlds, Online, vol. 21, no. 2, pp. 73-7, March-April 1997 discloses an information system that offers "periodical and newspaper content covering a wide range of business, news, and professional topics . . . letting the user search both text and graphics and build the product to suit. Articles can be retrieved in varying levels of detail: citation, abstracts, full text, and text with graphics. Images come in two flavors: Page Image, a virtual photocopy, and Text+Graphics, in which graphics are stored separately from the text and are manipulable as discrete items . . . . [The system] comes in two versions: Windows and Web."

John Mills Dudley, *Network-Based Classified Information Systems*, AU-A-53031/98 (27/08/98) discloses a "system for automatically creating databases containing industry, service, product and subject classification data, contact data, geographic location data (CCG-data) and links to web pages from HTML, XML, or SGML encoded web pages posted on computer networks such as Internets or Intranets . . . . The . . . databases may be searched for references (URLs) to web pages by use of enquiries which reference one or more of the items of the CCG-data. Alternatively, enquiries referencing the CCG-data in the databases may supply contact data without web page references. Data duplication and coordination is reduced by including in the web page CCG-data display controls which are used by web browsers to format for display the same data that is used to automatically update the databases."

Cordell et al, Automatic Data Display Formatting with A Networking Application, U.S. Pat. No. 5,845,084 (Dec. 1, 1998) discloses a placeholder image mechanism. "When a data request is made, the data transfer rate is monitored. When the receive data transfer rate is slow, and the data contains an embedded graphical image of unknown dimensions, a small placeholder image is automatically displayed for the user instead of the actual data. The small placeholder image holds a place on a display device for the data or the embedded graphical image until the data or embedded graphical image is received. When embedded graphical image is received, the placeholder image is removed, and the display device is reformatted to display the embedded graphical image."

Jonathon R. T. Lewis, System For Substituting Tags For Non-Editable Data Sets In Hypertext Documents And Updating Web Files Containing Links Between Data Sets Corresponding To Changes Made To The Tags, U.S. Pat. No. 5,355,472 (Oct. 11, 1994) discloses a "hypertext data processing system wherein data sets participating in the hypertext document may be edited, the data processing system inserting tags into the data sets at locations corresponding to the hypertext links to create a file which is editable by an editor and the data processing system removing the tags, generating a revised data set and updating the link information after the editing process. Its main purpose is to preserve the linking hierarchy that may get lost when the individual data sets get modified."

Wistendahl et al, System for Mapping Hot Spots in Media Content Interactive Digital Media Program, U.S. Pat. No. 5,708,845 (Jan. 13, 1998) discloses a "system for allowing media content to be used in an interactive digital media (IDM) program [that] has Frame Data for the media content and object mapping data (N Data) representing the frame addresses and display location coordinates for objects appearing in the media content. The N Data are maintained separately from the Frame Data for the media content, so that the media content can be kept intact without embedded codes and can be played back on any system. The IDM program has established linkages connecting the objects mapped by the N Data to other functions to be performed in conjunction with display of the media content. Selection of an object appearing in the media content with a pointer results in initiation of the interactive function. A broad base of existing non-interactive media content, such as movies, videos, advertising, and television programming can be converted to interactive digital media use. An authoring system for creating IDM programs has an object outlining tool and an object motion tracking tool for facilitating the generation of N Data. In a data storage disk, the Frame Data and the N Data are stored on separate sectors. In a network system, the object mapping data and IDM program are downloaded to a subscriber terminal and used in conjunction with presentation of the media content."

Rogers et al, Method for Fulfilling Requests of A Web Browser, U.S. Pat. No. 5,701,451 (Dec. 23, 1997) and Lagarde et al, Method for Distributed Task Fulfillment of Web Browser Requests, U.S. Pat. No. 5,710,918 (Jan. 20, 1998) disclose essentially "improvements which achieve a means for accepting Web client requests for information, obtaining data from one or more databases which may be located on multiple platforms at different physical locations on an Internet or on the Internet, processing that data into meaningful information, and presenting that information to the Web client in a text or graphics display at a location specified by the request."

Tyan et al, HTML Generator, European Patent Application No. EP 0843276 (May 20, 1998) discloses "generating an HTML file based on an input bitmap image, and is particularly directed to automatic generation of an HTML file, based on a scanned-in document image, with the HTML file in turn being used to generate a Web page that accurately reproduces the layout of the original input bitmap image."

TrueSpectra has a patent pending for the technology employed in its two products, IrisAccelerate and IrisTransactive. These products are designed for zooming and panning and simple image transformations and conversions, respectively. They support 10 file formats and allow developers to add new file formats via their SDK. They do not require the use of Flashpix for images. However, their documentation points out that performance is dependent on the Flashpix format. The system would be very slow if a non-Flashpix format was used.

TrueSpectra allows the image quality and compression to be set for JPEGs only. The compression setting is set on the server and all images are delivered at the same setting.

TrueSpectra has a simple caching mechanism. Images in the cache can be cleared out automatically at certain times and it does not have any dependency features for image propagation. The Web server needs to be brought down in order to update any original assets.

TrueSpectra does not require plug-ins to operate features such as zooming/panning or compositing. The alternative to plug-ins is using their Javascript or active server page technology. These technologies are used by many Web sites to provide interactivity, but not all Web browsers work correctly with these technologies.

TrueSpectra relies on Flashpix as its native file format and does not support media types such as multi-GIFs and sound formats. Flashpix files are typically larger than most file formats. Access to files is faster for zooming and panning, but appears to be quite slow.

The key to IrisTransactive is the compositing subsystem. It requires three things to build a shopping solution using image composition.

1) The original images must be created. It is suggested that the image be converted to Flashpix for better performance.
2) All of the individual images must be described in XML using the image composer program. The program allows the editor to specify anchor points, layer attributes, and layer names. The resulting file is between 5 k and 50 k.
3) The Web designer must place HTML referring to the XML in the Web site. By specifying parameters to the XML, the Web designer can turn on or off layers.

The herein above process for compositing images enables Web designers to create shopping sites. However, a lot of overhead is the result. The XML documents add 5 k-50 k to a Web site. The compositing commands that are embedded in the HTML are difficult to understand. And, because the compositing feature requires several steps to implement, it is not suitable for every image on a Web site. The process seems to be designed for the specific purpose of shopping.

MediaBin™ is limited to activities behind the firewall automating only the "post-creative busywork." In addition, MediaBin requires the use of an application server to function through a web interface. Thus images may not be directly added to any existing web page.

Macromedia's Generator operates by embedding variables in their proprietary Flash format. Therefore the actual imaging operations are somewhat limited and cannot be controlled directly from a web page request.

MGI Software sells point solutions that require end-users to download a viewer to process a proprietary image format.

PictureIQ offers a server-side image-processing appliance that provides a limited set of Photoshop functionalities. This appliance runs on the web-page server, processes information embedded in the web page, and rewrites the web page with image data.

The disclosed prior art fail to provide systems and methodologies that result in a quantum leap in the speed with which they can modify and add images, video, and sound to sites, in the volume of data they can publish internally and externally, and in the quality of the output. The development of such an automated media delivery system would constitute a major technological advance.

It would be advantageous to empower an end user with flexibility and control by providing interactive page capabilities.

It would be advantageous from an end user's perspective to generate Web pages that contain active graphics. For example, clicking on a Corvette image will cause a simple menu to pop up suggesting alternative colors and sizes in which to see the car. Clicking on portions of the image, such as a fender, can call up a close-in view of the fender.

It would be advantageous to provide an automated graphics delivery system that becomes part of the Web site infrastructure and operates as part of the Web page transaction and that thereby provides a less expensive and less time-consuming process.

It would be advantageous to provide a system for automated processing and delivery of media (images, video, and sound) to a Web server whereby it eliminates the laborious post-production and conversion work that must be done before a media asset can be delivered on a Web server.

It would be advantageous to create a dynamic Web site, wherein images are generated on demand from original assets, wherein only the original assets need to be updated, and wherein updated changes propagate throughout the site.

It would be advantageous to provide a system that generates media based on current Web server traffic thereby optimizing throughput of the media through the Web server.

It would be advantageous to provide a system that generates media that is optimized for the Web client, wherein client connection speed determines optimum quality and file size.

It would be advantageous to provide a system that generates media, whereby the media is automatically uploaded.

It would be advantageous to provide a system that automatically caches generated media so identical requests can be handled without regeneration of images.

It would be advantageous to provide a system that resides behind the Web server, thereby eliminating security issues.

It would be advantageous to provide a system wherein the client browser does not require a plug-in.

It would be advantageous to provide a system wherein the system does not require any changes to a Web server.

It would be advantageous to provide a system wherein the system manages the Web server media cache.

It would be advantageous to provide a system wherein the Web media is generated only if requested by a client browser.

It would be advantageous for a system to reduce the need for a Web author to create different versions of a Web site, the system automatically handling image content.

It would be advantageous to provide dynamic imaging capabilities, have a more complete set of image processing functionality, and be controlled directly through an image URL.

It would be advantageous to provide an end-to-end solution requiring only a standard browser that is completely controllable using the proprietary tags contained within a simple image link in the web page.

It would be advantageous to run an image application as a separate server controlled directly by single image requests to that server, such that any web server, even one that is only sending static HTML can access imaging features.

SUMMARY OF THE INVENTION

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, generated media is cached such that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a schematic diagram showing the placement of the system within a current Web infrastructure according to the invention;

FIG. **2** is a schematic diagram showing how a typical Web site delivers an HTML document and its graphics to a Web browser according to the prior art;

FIG. **3** is a schematic diagram showing delivery of an HTML document and media to a Web browser according to the invention;

FIG. **4** is a schematic diagram showing the components involved in Web site administration according to the prior art;

FIG. **5** is a schematic diagram showing the components of the system involved in Web site administration according to the invention;

FIG. **6** is a simple overview showing the components of the system according to the invention;

FIG. **7** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to the invention;

FIG. **8** is a flow chart showing an authoring process according to the invention;

FIG. **9** is a flow chart showing an HTML parsing process according to the invention;

FIG. **10** is a flow chart showing a media creation process according to the invention;

FIG. **11** is a screen shot showing an administration tool according to the invention;

FIG. **12** displays a structure of a database record used for the system according to the invention;

FIG. **13** shows original media to be processed according to the invention;

FIG. **14** shows a portion on an HTML document with a proprietary tag according to the invention;

FIG. **15** shows an HTML document and an HTML document source according to the invention;

FIG. **16** shows a generated GIF image according to the invention;

FIG. **17** is a schematic diagram of an image system within a typical Web infrastructure according to the invention;

FIG. **18** is a schematic diagram showing delivery of an HTML document and original media according to the invention;

FIG. **19** is a schematic diagram showing components of Web site administration according to a preferred embodiment of the invention;

FIG. **20** is a simple overview showing components of the image system according to a preferred embodiment of the invention;

FIG. **21** is a schematic diagram showing process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention;

FIG. **22** shows a flowchart of a content generation procedure according to a preferred embodiment of the invention; and

FIG. **23** is a flow chart showing an authoring process according to a preferred embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided.

The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, the generated media is cached so that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

A detailed description of such automatic media delivery system operating in parallel with existing Web site infrastructure is found below in the section under the heading as such.

FIG. **1** is a schematic diagram showing the placement of the system within a current Web infrastructure according to a preferred embodiment of the invention. The system **100** is attached to a Web server **110**, which is connected to multiple client browsers **120**(*a-d*) via the Internet **130**.

FIG. **2** is a schematic diagram showing how a typical Web site delivers an HTML document and its graphics to a Web browser according to the prior art. An original media **200** is passed to post-production systems **210**, wherein the media **200** is manipulated by hand and prepared for the Web. The result is a Web media **220**. The Web media **220** and an associated HTML document **230** referring to the media **220** by media tags are input to a Web server **110** for a Web browser **120** to view via the Internet **130**.

FIG. **3** is a schematic diagram showing delivery of an HTML document and media to a Web browser according to a preferred embodiment of the invention. An original media **200** and an HTML document embedded with proprietary media tags **300** are input into the system **100**. The system **100** generates a Web-safe media **220** and a modified HTML document **230** that refers to the Web media, and automatically loads them onto the Web server **110** for view by a Web browser **120** via the Internet **160**.

FIG. **4** is a schematic diagram showing components involved in Web site administration according to the prior art. Original media assets **400** are original images, video, or sound that have not been prepared for the Web. Web sites usually need to manage the placement of media on the network for easy retrieval by Web designers. Post-production systems **410** vary from Web site to Web site. Post-production systems **410** are usually custom procedures that Web designers use to convert an original media, such as an image, to one that can be displayed on the Web. Post-production systems **410** also upload finished images to Web image systems. Web images **420** are Web versions of the original images. Web images **420** are ready for retrieval by the Web server **110** to be delivered to a Web browser **120**. Any image to be modified or updated must pass through the herein above three components before it can be delivered to the Web browser **120**. HTML pages **460** have references to Web images **420**.

FIG. **5** is a schematic diagram showing the components involved in Web site administration according to a preferred embodiment of the invention. Web site administration is simplified using the claimed invention. Asset management, automatic image manipulation, automatic image conversion, automatic image upload, and automatic disk management **500** are provided by the claimed invention.

FIG. **6** is a simple overview showing the components of the system according to a preferred embodiment of the invention. HTML with proprietary tags **300** is the original HTML document that is embedded with proprietary tags which describe how the images are to be manipulated for the Web. Java servlet engine **600** is a third-party product that allows the system **100** to interface with the Web server **110** and execute Java servlet code. The Web server **110** is third-party software that delivers Web pages to a Browser **120**. The Browser **120** views Web pages that are sent from the Web server **110**. Modified HTML with system created images **230** are a final result of the system. Modified HTML **230** is a standard HTML document without proprietary embedded tags and with standard Web graphics.

The System.

A preferred embodiment of the system **100** is provided.

HTML parsing subsystem **610** parses through an HTML document and searches for proprietary tags. If it finds a proprietary tag it hands it to a media caching subsystem **620** for further processing. The media caching subsystem **620** returns a standard HTML tag. The HTML parsing subsystem **610** then replaces the proprietary tag it found with the returned tag. The parsing subsystem **610** then continues searching for a next proprietary tag, repeating the process herein above. The process is finished when no more proprietary tags can be found.

The media caching subsystem **620** determines if an image has been created for the requested proprietary tag. If the image has already been created and the files that built that image have not been modified, the media caching subsystem **620** returns an HTML tag that refers to a previously-generated image. If the image has not been created, the media caching subsystem **620** hands the HTML tag to a media creation subsystem **630**. The media creation subsystem **630** returns an image to the media caching subsystem **620**. The media caching subsystem **620** adds the created image and the HTML tag to a media cache database **640**.

The media cache database **640** contains references to the created images **645**. In a preferred embodiment, the references are the script used to create the image, the names of the images used to create the image, the dates of those files, and the HTML that represents the created image. The media caching subsystem **620** performs lookups in this database to determine if the image has been created. If the image has not been created the media caching subsystem **620** calls upon the media creation subsystem **630** to create the image and then store the results in the media cache database **640**.

The media creation subsystem **630** takes a proprietary tag from the media caching subsystem **620** and generates an image. The image is generated by deciphering the tag and handing it to the media processing engine **650**. After the image is created, the media creation subsystem returns the name of the newly created image to the media caching subsystem **620**.

The media processing engine **650** interprets the proprietary tag and generates the image. The media processing engine **650** looks up images in a media repository to obtain the location of the original file.

The media repository **660** contains original images **665** used in the system **100**.

FIG. **7** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention. An original media **200** is created. The media **200** is placed into the system **100** in the media repository **660**. Similarly, an HTML document with proprietary tags **300** is created and placed on a Web server **110**. A user requests a Web page from a Web browser **120**. The Web server **110** passes the requested page to an HTML parser **610**. The HTML parser **610** parses HTML looking for media tags. The parser **610** looks up media tags in a media tags database **640**. If the media tag is found, then the system **100** produces a modified HTML document **230**. Otherwise, the media creation subsystem **630** uses the media tag to generate a Web media **220**. The generated Web media **220** is placed in a media cache subsystem **620**. The proprietary media tag is converted by a converter **700** to a standard HTML tag that refers to the generated media **220** in cache. The media tag and the HTML equivalent are stored in the media tags database **640**. Media tags are replaced by standard HTML equivalent to provide a modified HTML document **230**. The modified HTML document **230** is delivered to the Web server **110**. The Web server **100** delivers the modified HTML document **230** to the browser **120** via the Internet for a user to view.

FIG. **8** is a flow chart showing an authoring process according to a preferred embodiment of the invention. The process starts (**800**) when a user adds an original graphic to the system (**810**). The user then creates an HTML document that contains proprietary media tags (**820**). The user then places the HTML document on a Web server (**830**) and ends the authoring process (**840**).

FIG. **9** is a flow chart showing an HTML parsing process according to a preferred embodiment of the invention. The process starts (**900**) when a consumer requests a Web page (**910**). A Web server hands the request of the Web page to the system (**920**). The system parses the Web page (**930**). The system looks for a media tag (**940**). If found, the system retrieves the HTML equivalent of the media tag (**950**) and replaces the media tag with the HTML equivalent tag (**960**). The system continues parsing the Web page for tags (**970**) by returning to step (**940**). When no more tags are found, the system delivers the modified Web page to the Web server (**980**) and therein ends the process (**990**).

FIG. **10** is a flow chart showing a media creation process according to a preferred embodiment of the invention. The process starts (**1000**) when the system requests an HTML equivalent to a proprietary media tag (**1010**). The Media tag is combined with bandwidth information (**1020**). The subsystem checks if the media tag already exists in the media tag database (**1030**). If it does, the subsystem checks if any of the

original assets used to create the media have been changed (**1040**). If not, then the subsystem retrieves the HTML equivalent tag from the database (**1050**) and returns the HTML equivalent tag to the requesting system (**1060**). If any of the original assets used to create the media have been changed (**1040**), then the subsystem removes the media tag entry from the media database (**1070**) and creates the media using the media tag (**1080**). The subsystem then stores the media in a media cache (**1090**). The subsystem generates the HTML referring to the generated media (**1100**) and places the media tag and the HTML equivalent in the media tag database (**1110**). The HTML equivalent is returned to the requesting system (**1060**) and the process stops (**1120**).

The differences between using HTML and the proprietary tags disclosed herein are noted. HTML allows Web designers to create Web page layouts. HTML offers some control of the images. HTML allows the Web designer to set the height and width of an image. However, all of the other image operations disclosed herein are supported by the claimed invention and are not supported by HTML.

Table A herein below provides the claimed proprietary tags according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

TABLE A

Tags
Generate image
<freerideimage> mediascript </freerideimage> Generate a standard Web image. Generate thumbnail image linked to full image
<freerideimagethumbnail> mediascript <xs=size ys=size /freerideimagethumbnail> Generate a thumbnail of specified size and link it to the full size version. Generate zoom and pan image
<freerideimagezoom> mediascript </freerideimagezoom> Generate a zoomable/panable image. Security
<freerideimagesecure> </freerideimagesecure> Specifies that all images found between these tags are secured images and the system will determine access before generating.

Table B herein below provides the claimed script commands according to a preferred embodiment of the invention. Additional commands may be added as needed.

TABLE B

Media processing script commands
Add Noise
Noise_AddNoise([amount=<value 1..999>] [gaussian] [grayscale]) This command adds noise to the image. Adjust HSB
AdjustHsb([hue @ <value ±255>] [saturation @ <value ±255>] [brightness @ <value ±255>]) This command allows the HSB of an image to be altered. This can be applied to images of all supported bit-depths. Adjust RGB
AdjustRgb([brightness @ <value ±255>] [contrast @ <value ±255>] [red @ <value ±255>] [green @ <value ±255>] [blue @ <value ±255>] [noclip @ <true, false>] [invert @ <true, false>]) This command allows the contrast, brightness, and color balance of an image to be altered.

TABLE B-continued

Media processing script commands
Blur
Blur(radius @ <value 0..30>) This command applies a simple blur filter on the image. Blur Convolve
Blur_Blur( ) This command commands perform a simple 3×3 convolution for blurring. Blur Convolve More
Blur_MoreBlur( ) This command commands perform a stronger 3×3 convolution for blurring. Blur Gaussian
Blur_GaussianBlur([radius=<value 0.1..250>]) This command applies a Gaussian blur to the image. Blur Motion
Blur_MotionBlur([distance=<value 1..250>] [angle=<degrees>]) This command applies motion blurring to the image using the specified distance and angle. Brush Composite
Composite(source @ {<User-Defined Media Object name>} [x @ <pixel>] [y @ <pixel>] [onto] [opacity @ <value 0..255>] [color @ <color in hexadecimal>] [colorize @ <true, false>] [saturation @ <value 0..255>]) This command composites the specified “brush” (foreground) image onto the current “target” (background) image. Colorize
Colorize(color @ <color in hexadecimal> [saturation @ <value 0..255>]) This command changes the hue of the pixels in the image to the specified color. Convert
Convert(rtype @ <bit-depth> {dither @ <value 0..10>]) This command converts the image to the specified type/bit-depth. Convolve
Convolve(Filter @ <filtername>) This command applies a basic convolution filter to the image. In a user interface driven system, the filters could be stored in files and edited/created by the user. Crop/Resize Canvas
Crop([xs @ {<pixels>, <percentage + “%”>}] [ys @ {<pixels>, <percentage + “%”>}] [xo @ <left pixel>] [yo @ <top pixel>] [padcolor @ <color in hexadecimal>] [padindex @ <value 0..255>]) This command crops the media to a specified size. Discard
Discard( ) This command removes the designated Media Object from memory. Drop Shadow
DropShadow([dx @ <pixels>] [dy @ <pixels>] [color @ <color in hexadecimal>] [opacity @ <value 0..255>] [blur @ <value 0..30>] [enlarge @ <true, false>]) This command adds a drop shadow to the image based on its alpha channel. Equal
Equal(source @ {<User-Defined Media Object name>}) This command compares the current media with the one specified. If the media are different in any way, an error value is returned. Equalize
Equalize([brightness @ <−1, 0..20>] [saturation @ <−1, 0..20>]) This command equalizes the relevant components of the media. Equalization takes the used range of a component and expands it to fill the available range. Export Channel
ExportGun(Channel @ <channelname>) This command exports a single channel of the source as a grayscale image. Find Edges
Stylize_FindEdges([threshold=<value 0..255>] [grayscale] [mono] [invert]) This command finds the edges of the image based on the specified threshold value. Fix Alpha
FixAlpha( ) This command adjusts the RGB components of an image relative to its alpha channel.

TABLE B-continued

Media processing script commands
Flip
Flip(<horizontal, vertical> @ <true, false>) This command flips the media vertically or horizontally. Frame Add
FrameAdd(Source @ <filename>) This command adds the given frame(s) to the specified Media Object. Glow/Halo
Glow(Size @ <value 0..30> [halo @ <value 0..size>] [color @ <color in hexadecimal>] [opacity @ <value 0..255>] [blur @ <value 0..30>] [enlarge @ <true, false>]) This command produces a glow or halo around the image based on the image's alpha. High Pass
Other_HighPass([radius=<value 0.1..250>]) This command replaces each pixel with the difference between the original pixel and a Gaussian blurred version of the image. Import Channel
ImportGun(channel @ <channel name> source @ {<User-Defined Media Object name>} [rtype @ <bit-depth>]) This command imports the specified source image (treated as a grayscale) and replaces the selected channel in the original. Load
Load(Name @ <filename> [type @ <typename>] [transform @ <true, false>]) This command loads a media from the specified file. Maximum
Other_Maximum([radius=<value 1..10>]) This command scans the area specified by the radius surrounding each pixel, and then replaces the pixel with the brightest pixel found. Minimum
Other_Minimum([radius=<value 1..10>]) This command scans the area specified by the radius surrounding each pixel, and then replaces the pixel with the darkest pixel found. Normalize
Normalize([clip @ <value 0..20>]) This command expands the volume of the sample to the maximum possible. Pixellate Mosaic
Pixellate_Mosaic([size=<value 2..64>]) This command converts the image to squares of the specified size, where each square contains the average color for that part of the image. Pixellate Fragment
Pixellate_Fragment([radius=<value 1..16>]) This command produces four copies of the image displaced in each direction (up, down, left, right) by the specified radius distance and then averages them together. Quad Warp
QuadWarp([tlx=<position>] [tly=<position>] [trx=<position>] [try=<position>] [blx=<position>] [bly=<position>] [brx=<position>] [bry=<position>] [smooth]) This command takes the corners of the source image and moves them to the specified locations, producing a warped effect on the image. Reduce to Palette
Reduce([colors @ <num colors>] [netscape @ <true, false>] [b&w @ <true, false>] [dither @ <value 0..10>] [dithertop @ <value 0..10>] [notbackcolor] [pad @ <true, false>]) This command applies a specified or generated palette to the image. Rotate
Rotate(Angle @ <value 0..359> [smooth @ <true, false>] [enlarge @ <true, false>] [xs @ <pixels>] [ys @ <pixels>]) This command rotates the media by the specified angle in degrees. Rotate 3D
Rotate3d([anglex @ <angle ±89>] [angley @ <angle ±89>] [distance @ <value>]) This command rotates the image in 3D about either the x-axis or y-axis. Save
Save([type @ <image-type>]) This command saves a media to the specified file.

TABLE B-continued

Media processing script commands
Scale
Scale([xs @ {<pixels>, <percentage + “%”>}] [ys @ {<pixels>, <percentage +“%”>}] [constrain @ <true, false>] [alg @ {“fast”, “smooth”, “outline”}] [x1 @ <pixels>] [y1 @ <pixels>] [x2 @ <pixels>] [y2 @ <pixels>]) This command scales the image to the specified size.
Select
Selection([source @ <User-Defined media Object>}] [remove @ <true, false>] [invert @ <true, false>] [backcolor] [color=<color>] [index=<value>] [opacity @ <value 0..255>]) This command manages the selected region for the current Media Object.
Set Color
SetColor([backcolor @ <color in hexadecimal>] [forecolor @ <color in hexadecimal>] [backindex @ <value 0..255>] [foreindex @ <value 0..255>] [transparency @ (“on”,”off”)]) This command allows the background color, foreground color, and transparency state of an image to be set.
Set Resolution
SetResolution([dpi @ <value>] [xdpi @ <value>] [ydpi @ <value>]) This command changes the DPI of the image in memory.
Sharpen
Sharpen_Sharpen( ) This command sharpens the image by enhancing the high-frequency component of the image.
Sharpen More
Sharpen_SharpenMore( ) This command sharpens the image by enhancing the high-frequency component of the image, but is stronger than the standard sharpening.
Stylize Diffuse
Stylize_Diffuse([radius=<value 0..>] [lighten] [darken]) This command diffuses the image by randomizing the pixels within a given pixel radius.
Stylize Embose
Stylize_Emboss([height=<value 1..10>] [angle=<degrees>] [amount=<percentage 1..500>]) This command converts the image to an embossed version.
Text Drawing
DrawText(Text @ <string> Font @ <font file> [size @ <value>] [color @ <color in hexadecimal>] [smooth @ <true, false>] [<left, right, top, bottom> @ <true, false>] [x @ <pixel>] [y @ <pixel>] [wrap @ <pixel-width>] [justify @ {left,center,right}] [angle @ <angle>]) This command composites the specified text string onto the image.
Text Making
MakeText(text @ <string> font @ <font file> [path @ <path to font directory>] [size @ <value 1..4095>] [color @ <color in hexadecimal>] [smooth @ <true, false>] [wrap @ <pixel-width>] [justify @ {left,center,right}] [angle @ <angle>]) This command creates a new image that includes only the specified text.
Trace Contour
Stylize_TraceContour([level=<value 0..255>] [upper] [invert]) This command traces the contour of the image at the specified level (for each gun).
Unsharpen Mask
Sharpen_UnsharpMask([amount=<percentage 1..500>] [radius=<value 0.1..250>] [threshold=<value 0..255>]) This command enhances the edges and detail of an image by exaggerating differences between the image and a gaussian blurred version of the same image.
Zoom
Zoom([xs @ <pixels>] [ys @ <pixels>] [scale @ <value>] [x @ <left pixel>] [y @ <top pixel>]) This command zooms in on a specified portion of the media and fits it to the specified size. This constitutes a crop followed by a scale.

Table C herein below provides a list of features provided by a preferred embodiment of the invention. It is noted that the list of features included in Table C is by no means complete. In other embodiments, the list of features is expanded or reduced as needed.

TABLE C

System Feature List
Reads and writes various file formats:
BMP, GIF, JPG, PNG, TIF, PICT, TGA, PSD, FPX;
Supports many image processing operations;
Dynamically creates Web images from original assets;
Dynamically creates thumbnail images;
Dynamically creates images that can be panned and zoomed without browser plug-ins or special file formats;
Automatically propagates changes of original assets throughout a Web site;
Uses an intelligent caching mechanism:
Clean up image cache on demand;
Eliminates orphaned image files; and
Optimizes Web server cache by providing most recent images;
Renders TrueType fonts on the server instead of browser;
Uses intelligent scaling of line drawings;
Allows Web designers to manipulate images with proprietary tags;
Preserves original image assets;
Optimizes Web server traffic by adjusting the bandwidth of graphics;
Optimizes images for client connection speed;
Allows clients to specify the quality of images on a Web site; and
Allows Web designers to dynamically create images by manipulating proprietary tags in their applications (server or client side).

FIG. **11** is a screen shot showing an administration tool according to a preferred embodiment of the invention. Specifically, FIG. **11** shows an administration page that is configured to contain cached images of generated scripts. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **12** displays a structure of a database record used for the system according to a preferred embodiment of the invention. A Script Table **1200** has 5 columns, Media Script **1210**, HTML Equivalent **1220**, Bandwidth **1230**, Generated File **1240**, and Dependency List **1250**. A Dependency Table **1260** has two columns, File Name **1270** and Modification Date **1280**.

Snowboard Store Example.

Background

The snowboard store highlights several features of the claimed system. The snowboard store is an imaginary store that allows a user to configure his or her snowboard. The store consists of five logos, five board colors, and four boards. The consumer clicks on the buttons to change the snowboard represented in the middle of the screen. When the consumer has configured the snowboard they the snowboard can be purchased by selecting a buy button.

Prior Art Method.

To create the snowboard site today, the Web designer must render all possible combinations of the board. The number of combinations is five logos×five board colors×four boards=100. The designer also must render all the buttons. The creation process is very tedious and involves a lot of production work. Typically, most Web sites do not even attempt such an endeavor. Also, other issues must be addressed, such as, for example, updating the Web site and scripting. For example, updating a single logo involves updating a minimum of 20 images.

The prior art method sustains a graphic intensive site that requires management of at least 100 images. Updates to the Web site are time-consuming and prone to human error.

The Claimed Method.

A preferred embodiment of the method scripts the image creation process in HTML to create a dynamic Web site. There is no need to create over 100 images. The claimed system generates images on demand. The Web site only needs to create original assets. The scripting process involves writing the proprietary scripts. In the current example herein, scripting buttons is very simple. Once one button is created, simply copy and paste the HTML to create another button or many buttons. Only the name of the image to be overlaid on the button must be changed. The Webmaster then creates a simple program that reads what object a user has clicked on and generates a proprietary tag. The tag is then sent to the claimed system to generate a center image.

The claimed method allows the creation of all 100 combinations automatically. When the Web site receives an updated image, only the original image needs to be updated. Any change to the original image automatically propagates throughout the system. The Web site is easier to manage. Testing of the Web site is easier because there is no need to test all 100 combinations. A small subset of combinations will guarantee adequate coverage.

Processing of an Image Tag Example (FIG. **13**-**16**).

FIG. **13** shows two original images **1300** and **1310** to be processed according to a preferred embodiment of the invention.

FIG. **14** shows a portion on an HTML document configured to include a proprietary tag **1400**, <freerideimage></freerideimage> according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **15** shows an HTML document **1500** as viewed in a browser and an HTML document source **1510**, according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **16** shows a generated GIF image **1600** according to a preferred embodiment of the invention.

Automatic Media Delivery System Operating in Parallel with Existing Web Site Infrastructure.

It should be noted that the words, media, graphics, and images are used herein interchangeably.

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, the generated media is cached so that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

A preferred embodiment of the invention is described with reference to FIG. **17**. FIG. **17** is a schematic diagram of an image system within a typical Web infrastructure according to the invention. The image system **100** is placed in parallel to an existing Web server **110**. The image system **100** may be on-site or off-site to the Web infrastructure. Multiple client browsers **120***a*-**120***d* communicate with both the Web server **110** and the image server **100** via the Internet **130**.

The delivery of an HTML document and media according to a preferred embodiment of the invention is described with reference to FIG. **18**. Resource locators (URLs) are placed within HTML documents **301** accessible to the Web server **110**. These URLs direct browsers to generate requests for media to the system **100**. The system processes such URLs by interpreting the proprietary tags, executing the indicated image generation procedures on the original media **200**, and returning derivative Web-safe media to client browsers **120***a*-**120***d* via the Internet **130**. Additionally, such generated media is cached on the image server **100** and, therefore need not be regenerated for subsequent requests.

Web site administration according to another preferred embodiment of the invention is described with reference to FIG. **19**. FIG. **19** is a schematic diagram showing the components of Web site administration according to a preferred embodiment of the invention, whereby Web site administration is simplified. The preferred embodiment provides, but is not limited to the following services: asset management, automatic image manipulation, automatic image conversion, automatic image upload, automatic image customization based on browser characteristics, automatic disk management, automatic control of proxy caching, and image delivery **501**.

FIG. **20** is a simple overview showing components of the system according to a preferred embodiment of the invention. HTML pages with proprietary URL tags **301** describe how referenced media therein is to be manipulated for Web. Browsers **120** send such tags to the image system **100** as media requests. A server **2000** within the image system **100** receives the media requests, decodes the URL tags, and retrieves any media that already exists in the media caching system **2010**. Non-existent media is subsequently generated by a media creation system **2020** using original media **2050** stored in a media repository **2040** and using content generation procedures **2030**.

The Image System.

Following is a detailed description of the preferred embodiment of the invention with reference to FIG. **21** below.

The system receives a request for media through a URL containing proprietary tags for controlling image generation. The system parses this URL to determine the content generation procedure to execute, input to the content generation procedure, post-processing directives for, for example, zoom/pan/slice, browser properties, and any cache control directives. Such data is handed to a media caching subsystem that returns the requested image if found. If the image is not found, the information is handed to the media generation subsystem that executes the specified content generation procedure to produce a derivative image. The media generation subsystem returns one or more images to the media caching system for subsequent reuse.

The media caching subsystem is a mechanism for associating final or intermediate derivative media with the procedure, input, and user characteristics used to generate said media, specified through proprietary tags within the requested URL. This system may be implemented using a database, file system, or any other mechanism having capability to track such associations.

The media generation subsystem executes a primary content generation procedure to produce a derivative image whose identifier is provided to the media caching subsystem. This derivative image is composed of one or more original images acquired from the media repository. This media is then passed to the dynamic image content system, if necessary, to generate a subsequent derivative media suitably modified for the needs of zooming, panning, or slice. The resulting media is passed to the user profile system where it is again modified to account for any specific user browser characteristics specified using the proprietary URL tags. This media is then returned to the browser, along with any cache control directives encoded within the URL, and its identifier is passed to the media caching system for subsequent retrieval.

The dynamic content system operates on intermediate derivative images to generate image subsets or scalings used by Web site designers to implement zooming in on an image, panning across an image, slicing an image into parts, and the like for special Web page effects. The input to this system is cached by the media caching system such that the intermediate image need not be regenerated.

The user profile system operates on the final image about to be returned to the browser and may modify the image to account for individual needs of Web site users. The designer of a site is able to implement freely custom post-processing of images to meet the specific needs of their clients.

FIG. **21** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention. Original media **200** is created and placed into the system **100** in a media repository **2040**. A content generation procedure **2140** is created with instructions on how the media is to be transformed to create the desired Web page content. An HTML page **301** is created for the Web site comprising the system **100**, the page containing one or more URLs directing a browser **120** to request the specified content generation procedure **2140** from the system **100** using input parameters specified with proprietary tags encoded within the URL. The browser **120** requests the Web page **301** from the Web site **110**. Upon receipt of the page **301**, the browser contacts the system **100** requesting media specified in the URL. The system parses the URL **2100** to determine the content generation procedure **2140** to execute, any corresponding input parameters to be used by such procedure, any dynamic content processing **2150** to be performed by dynamic media procedures, any user profile information **2160** to be used to modify the resulting image, and any cache control HTTP headers **2190** the site instructs to accompany the resulting image.

The parser generates a unique primary lookup key **2110** for the specified resulting media. If the key corresponds to an existing generated media **2180**, such media is returned immediately to the browser **120** through a media cache **2120**, and the transaction is complete. Otherwise, a media generation occurs. In the case of media generation requiring dynamic content processing, a unique secondary lookup key corresponding to intermediate media is generated **2130**. If intermediate media **2170** corresponding to this key is found, such media is passed directly to the dynamic media content system **2150** having dynamic media procedures, wherein appropriate action is taken to generate the required derivative from the intermediate media data. A unique key is generated for the derivative **2130** and passed to the media caching system **2120**. If the media caching system finds no such intermediate image, such intermediate image is generated according to instructions specified by the content generation procedure, cached by the media cache system **2120** as a secondary cached media **2170**, and passed to the dynamic media system

2150. Again, appropriate action is taken to generate the required derivative from the intermediate image data.

The resulting image after any dynamic media processing is complete, is checked to ensure that the image is in a valid Web image format. If not, the image is automatically converted into a valid format.

The final media is passed to a user profile system **2160** wherein browser characteristics specified through proprietary tags within the URL are inspected, and appropriate modification to the media is performed, based on such characteristics. The resulting image is handed to the media cache system **2120** for caching and returned to the browser **120**.

FIG. **22** shows a flowchart of the content generation procedure according to a preferred embodiment of the invention. A URL containing proprietary tags (**2200**) is parsed (**2210**) to determine the content generation procedure to execute, any dynamic modifications to the media, user profile characteristics, and proxy-cache control. A unique final lookup key is generated for the media (**2220**) and the media cache is checked (**2230**). If the indicated media exists, control passes to proxy-cache control (**2290**) and the media is delivered to the browser (**2295**). Otherwise, dynamic media system tags are separated from content generation control tags (**2240**) and a unique intermediate image lookup key is generated (**2250**). The cache is then checked for such intermediate media (**2261**). If such intermediate media is found, it is used directly for dynamic processing, if required. Otherwise, content is generated (**2262**) and cached (**2263**), and the result is evaluated for dynamic processing (**2270**). If dynamic processing is required, the media is operated upon by the dynamic content generator (**2271**), otherwise it is evaluated for valid content type (**2272**). If the content type is invalid, the media is automatically converted to a valid type (**2273**). The resulting image is then customized by the user profiling system (**2280**) for specified browser or client attributes. Finally, any cache-control directives specified are attached to the response (**2290**) and the media is delivered to the browser (**2295**).

FIG. **23** is a flow chart showing an authoring process according to a preferred embodiment of the invention. The process starts (**2300**) when a user adds an original graphic or other media (**2310**) to the system. The author then creates a content generation procedure (**2320**) containing instruction on how the original media should be processed to generate the desired Web page content. The user then creates an HTML document (**2330**) that refers to that image by using a URL pointing to a content generation procedure on the image server. The system ends (**2340**). The authoring subsystem assists the Web site designer with choosing parameters and with designing the content generation procedure such that the desired Web site graphic is obtained.

It should be appreciated that differences exist between specifying an image with a URL and requesting an image using a content creation process that interprets proprietary parameters encoded within a URL. That is, URLs allow Web site designers to load specific graphic images into a Web page. In contrast and according to the invention, URLs containing proprietary content creation tags initiate a process whereby graphic images for a site are automatically produced.

Table D below is a list of example proprietary URL tags used for content generation within the system according to the preferred embodiment of the invention. Additional tags may be added to the system as necessary.

TABLE D

Tags
f = function Names the content creation procedure used to generate all or part of the desired graphic.
args = arguments Supplies page dependent parameters used to control the content creation procedure from within the Web page.
cr = crop rectangle Specifies that portion of the image generated by the content generation procedure to be returned to the browser.
st = slice table Specifies a rectangular grid to be placed over the image produced by the content generation procedure, each portion of which can be returned to the browser.
sp = slice position Specifies that portion of the slice table grid placed over the image generated by the content creation procedure to be returned to the browser.
is = image size parameter Specifies scale factors to be applied to any portion of an image generated by any combination of a content generation procedure, arguments, crop rectangles, slice tables, and slice positions.
p = user profile string Specifies a user profile identifier used to modify the final image prior to returning the image to the browser, thus allowing clients to modify the image returned to the browser to account for individual browsing conditions.
c = cache control Specifies a proxy-cache control string to accompany the returned image within an HTTP header.

Table E below is a list of example supported content creation commands according to a preferred embodiment of the invention. Additional commands may be added as necessary.

TABLE E

Content Creation Commands
Adjust HSB Allows the HSB of an image to be altered.
Adjust RGB Allows the contrast, brightness, and color balance of an image to be altered.
Colorize Alters the hue of the pixels in the image to that of the specified color.
Brush Composite Composites the specified brush image onto the current target image.
Convert Converts the rasters to the specified type/bit-depth.
Crop Crops the media to the specified size.
Dropshadow Adds a drop shadow to the image, based on the alpha-channel.
Equalize Performs an equalization on the relevant components of the media.
FixAlpha Adjusts the RGB components of an image relative to its alpha-channel.
Flip Flips the media vertically or horizontally.
Glow Produces a glow or halo around the image.
Load Loads in a media from the specified file.
Normalize Similar to equalize, but for audio.
Reduce Reduces the image to a specified palette.
Rotate Rotates the media clockwise by the specified angle in degrees.
Save Saves the media to the specified file.
Scale Scales the media to the specified size.

TABLE E-continued

Content Creation Commands
SetColor Allows the background color, foreground color, and transparency state of the media to be set.
Text Drawing Composites the specified text onto the image.
Text Making This command, instead of compositing text onto the target, creates a new image that just encloses the text.
Zoom Zooms in on a specified portion of the media, and fits it to the specified size. Effectively this constitutes a crop followed by a scale.

Table F below lists comprises some, and is not limited to all major features of a preferred embodiment of the invention. Additional features may be added as necessary.

TABLE F

System Features
Reads and writes various file formats;
Supports many image processing operations;
Dynamically creates Web images from original assets;
Dynamically creates thumbnail images;
Dynamically and efficiently creates images that can be panned, zoomed, or sliced from original assets without Browser plugins;
Automatically propagates changes in original assets throughout the Web site;
Uses an intelligent caching mechanism for both final and intermediate graphics, comprising:
Clean up cache on demand;
Eliminates orphaned Web files; and
Optimizes Web server cache by providing most recent images;
Renders True-Type fonts on server instead of browser;
Uses intelligent scaling of line drawings;
Allows Web designers to manipulate images using a combination of content generation procedures and proprietary URL tags;
Preserves original image assets;
Optimizes Web server traffic by adjusting the bandwidth of graphics;
Optimizes images for client connection speed;
Allows clients to specify the quality of images on a Web site;
Allow site-specific customized image optimizations for a variety of purposes; and
Allows Web designers to dynamically create images by manipulating proprietary URL tags in applications.

Accordingly, although the invention has been described in detail with reference to a particular preferred embodiment, persons possessing ordinary skill in the art to which this invention pertains will appreciate that various modifications and enhancements may be made without departing from the spirit and scope of the claims that follow.

The invention claimed is:

1. A method for providing simultaneous transcoding of multi-media data, comprising:

receiving a multi-media file in a first format;

receiving a request with proprietary tags containing parameters as part of a URL for said multi-media file to a plurality of output devices, wherein at least two devices from said plurality of output devices output file formats with incompatible capabilities;

automatically determining, at the time of the request, one or more formats requested based on said one or more proprietary tags,

parsing said parameters to determine client connection speed, server traffic, browser, and device based upon said client connection speed, said server traffic, polled device, browser, and other variables collected at the time of said request, and one or more content, generation procedures, automatically transcoding, at the time of the request and without input by a network administrator, data within said multi-media file into a plurality of formats based on one or more formats of any output device from among the plurality of output devices, wherein said plurality of formats are different from said first format wherein said transcoding comprises each of: changing a resolution of said data within said multi\-media file based on the resolution capabilities of the at least two output devices, changing a frame rate of said data within said multi-media file based on the frame rate capabilities of the at least two output devices, and changing a compression format of said data within said multi-media file based on the compression capabilities of the at least two output devices; and

wherein said first format is an analog format and wherein said at least one alternate format comprises a first digital format and a second digital format; and

transmitting simultaneously with said transcoding said multi-media file to each output device from among said plurality of output devices in a compatible format.

2. The method of claim **1**, wherein said data within said multi-media file comprises at least one of:

video data, audio data or digital pictures.

3. The method of claim **1**, wherein said transmitting said multi-media file to an output device comprises transmitting said multi-media file to a display device.

4. The method of claim **1**, wherein said transmitting said multi-media file to an output device comprises transmitting said multi-media file to a storage device.

5. The method of claim **1**, further comprising:

transmitting said transcoded multi-media file in said alternate format to an external device.

6. The method of claim **5**, wherein said external device comprises a portable media player.

7. The method of claim **1**, wherein said multi-media file in said first format is received from a service provider.

8. The method of claim **1**, wherein said multi-media file in said first format is received from a local content source.

9. A system for providing simultaneous transcoding of multi-media data, comprising:

a controller for receiving a multi-media file in a first format and, responsive to a request with proprietary tags containing parameters as part of a URL for said multi-media file, for transmitting said multi-media file to an output device from among a plurality of output devices, wherein at least two output devices from said plurality of output devices output file formats with incompatible capabilities; and

a transcoder, coupled to said controller and said request parser, for parsing said parameters to determine client connection speed, server traffic, browser, and device based upon said client connection speed, said server traffic, polled device, browser, and other variables collected at the time of said request, and one or more content generation procedures, and for automatically transcoding, at the time of said request, data within said multi-media file into a plurality of formats based on one or more formats of any output device from among the plurality of output devices, wherein said plurality of output devices are different from said first format, while said multi-media file is transmitted to said output device, wherein said transcoder is configured for:

changing a resolution of said data within said multi-media file based on the resolution capabilities the at least two output devices, changing a frame rate of said data within said multi-media file based on the frame rate capabilities the at least two output devices, and

changing a compression format of said data within said multi-media file based on the compression capabilities the at least two output devices; and

wherein said first format is an analog format and wherein said at least one alternate format comprises a first digital format and a second digital format.

10. The system of claim **9**, wherein said data within said multi-media file comprises at least one of:

video data, audio data or digital pictures.

11. The system of claim **9**, wherein said output device comprises a display device.

12. The system of claim **9**, wherein said output device comprises a storage device.

13. The system of claim **9**, further comprising:

one or more interfaces for transmitting said transcoded multi-media file in said alternate format to an external device.

14. The system of claim **13**, wherein said external device comprises a portable media player.

15. A non-transitory computer-readable medium having stored thereon a plurality of instructions, the plurality of instructions including instructions which, when executed by a processor, cause the processor to perform the steps of a method for providing simultaneous transcoding of multi-media data, comprising:

receiving a multi-media file in a first format;

receiving a request with proprietary tags containing parameters as part of a URL for said multi-media file and transmitting said multi-media file to an output device; and

automatically determining, at the time of the request, one or more formats requested based on said one or more proprietary tags,

parsing said parameters to determine client connection speed, server traffic, browser, and device based upon said client connection speed, said server traffic, polled device, browser, and other variables collected at the time of said request, and one or more content generation procedures, and automatically transcoding, at the time of the request, data within said multi-media file into a plurality of formats based on one or more formats of any output device from among the plurality of output devices, wherein said plurality of formats are different from said first format, wherein said transcoder is configured for changing each of: a resolution of said data within said multi-media file based on resolution capabilities of at least two output devices, a frame rate of said data within said multi-media file based on frame rate capabilities of at least two output devices, and a compression format of said data within said multi-media file based on compression capabilities of at least two output devices;

wherein said first format is an analog format and wherein said at least one alternate format comprises a first digital format and a second digital format;

transmitting simultaneously with said transcoding said multi-media file to each output device from among said plurality of output devices in a compatible format.

16. The non-transitory computer readable medium of claim **15**, wherein said transmitting said multi-media file to an output device comprises transmitting said multi-media file to at least one of: a display device or a storage device.

17. The non-transitory computer readable medium of claim **15**, further comprising:

transmitting said transcoded multi-media file in said alternate format to an external device.

* * * * *

Exhibit F

US009158745B2

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

(10) **Patent No.: US 9,158,745 B2**
(45) **Date of Patent: Oct. 13, 2015**

(54) **OPTIMIZATION OF MEDIA CONTENT USING GENERATED INTERMEDIATE MEDIA CONTENT**

(71) Applicant: **AUTOMATED MEDIA PROCESSING SOLUTIONS INC.**, Sausalito, CA (US)

(72) Inventors: **Sean Barger**, Mill Valley, CA (US); **Brian Rice**, Darien, IL (US); **Matt Butler**, Beaverton, OR (US); **David Pochron**, Cambridge, MA (US)

(73) Assignee: **EQUILIBRIUM**, Sausalito, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 273 days.

(21) Appl. No.: **13/752,110**

(22) Filed: **Jan. 28, 2013**

(65) **Prior Publication Data**

US 2013/0138774 A1 May 30, 2013

Related U.S. Application Data

(60) Continuation of application No. 12/238,842, filed on Sep. 26, 2008, now Pat. No. 8,381,110, which is a division of application No. 12/173,747, filed on Jul. 15, 2008, now Pat. No. 8,656,046, which is a division

(Continued)

(51) **Int. Cl.**

G06F 17/00	(2006.01)
G06F 17/22	(2006.01)
G06F 17/21	(2006.01)

(Continued)

(52) **U.S. Cl.**
CPC ***G06F 17/2264*** (2013.01); ***G06F 17/211*** (2013.01); ***G06F 17/2235*** (2013.01); ***G06F 17/27*** (2013.01); ***G06F 17/30017*** (2013.01); ***G06F 17/30038*** (2013.01); ***G06F 17/30058*** (2013.01); ***G06F 17/30905*** (2013.01)

(58) **Field of Classification Search**
CPC ... G06F 17/211; G06F 17/2264; G06F 17/27; G06F 17/2235; G06F 17/30017; G06F 17/30058; G06F 17/30038
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,088,052	A	2/1992	Spielman et al.
5,355,472	A	10/1994	Lewis

(Continued)

FOREIGN PATENT DOCUMENTS

AU	5303198	8/1998
EP	0747842 A1	12/1996

(Continued)

OTHER PUBLICATIONS

"Tables of Contents service for Computers & Geosciences", Computers and GeoSciences, vol. 23, Issue 5; http://library. iem.ac. ru/comp&geol00983004/sz977014.html,, retrieved on Mar. 18, 2004 from website:, 1997, 2, pp.

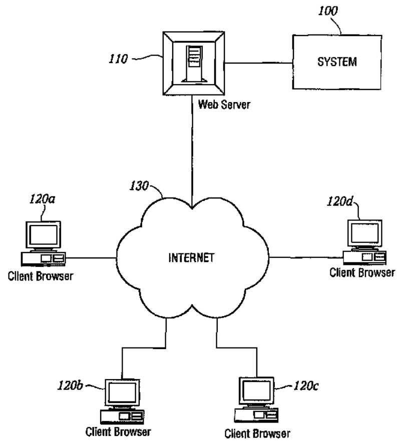
(Continued)

Primary Examiner — Cesar Paula
Assistant Examiner — David Faber
(74) *Attorney, Agent, or Firm* — Michael A. Glenn; Perkins Coie LLP

(57) **ABSTRACT**

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed by an author within URLs embedded within Web documents.

14 Claims, 23 Drawing Sheets



Related U.S. Application Data

of application No. 11/269,916, filed on Nov. 7, 2005, now abandoned, which is a continuation-in-part of application No. 09/929,904, filed on Aug. 14, 2001, now Pat. No. 6,964,009, which is a continuation-in-part of application No. 09/425,326, filed on Oct. 21, 1999, now Pat. No. 6,792,575.

(51) **Int. Cl.**

G06F 17/30	(2006.01)
G06F 17/27	(2006.01)

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,442,771	A		8/1995	Filepp et al.	
5,530,852	A		6/1996	Meske, Jr. et al.	
5,701,451	A		12/1997	Rogers et al.	
5,708,845	A		1/1998	Wistendahl et al.	
5,710,918	A		1/1998	Lagarde et al.	
5,737,619	A		4/1998	Judson	
5,740,430	A	*	4/1998	Rosenberg et al.	1/1
5,745,908	A		4/1998	Anderson et al.	
5,758,110	A		5/1998	Boss et al.	
5,761,655	A		6/1998	Hoffman	
5,793,964	A		8/1998	Rogers et al.	
5,819,261	A		10/1998	Takahashi et al.	
5,822,436	A		10/1998	Rhoads	
5,845,084	A		12/1998	Cordell et al.	
5,845,279	A		12/1998	Garofalakis et al.	
5,845,299	A		12/1998	Arora et al.	
5,860,068	A		1/1999	Cook	
5,860,073	A		1/1999	Ferrel et al.	
5,861,881	A		1/1999	Freeman et al.	
5,862,325	A		1/1999	Reed et al.	
5,864,337	A		1/1999	Marvin	
5,870,552	A		2/1999	Dozier et al.	
5,880,740	A		3/1999	Halliday et al.	
5,890,170	A		3/1999	Sidana	
5,895,476	A		4/1999	Orr et al.	
5,895,477	A		4/1999	Orr et al.	
5,903,277	A		5/1999	Sutherland et al.	
5,903,892	A		5/1999	Hoffert et al.	
5,937,160	A		8/1999	Davis et al.	
5,943,680	A		8/1999	Shimizu et al.	
5,956,737	A		9/1999	King et al.	
6,009,436	A		12/1999	Motoyama et al.	
6,167,442	A		12/2000	Sutherland et al.	
6,173,316	B1	*	1/2001	De Boor et al.	709/218
6,300,947	B1	*	10/2001	Kanevsky	715/866
6,310,601	B1	*	10/2001	Moore et al.	345/660
6,311,185	B1		10/2001	Markowitz et al.	
6,345,279	B1	*	2/2002	Li et al.	1/1
6,412,008	B1	*	6/2002	Fields et al.	709/228
6,456,305	B1	*	9/2002	Qureshi et al.	715/800
6,463,445	B1		10/2002	Suzuki et al.	
6,483,851	B1		11/2002	Neogi	
6,484,149	B1		11/2002	Jammes et al.	
6,539,420	B1	*	3/2003	Fields et al.	709/206
6,563,517	B1		5/2003	Bhagwat et al.	
6,591,280	B2		7/2003	Orr et al.	
6,623,529	B1		9/2003	Lakritz	
6,658,462	B1	*	12/2003	Dutta	709/219
6,789,170	B1	*	9/2004	Jacobs et al.	711/133
6,857,102	B1	*	2/2005	Bickmore et al.	715/205
6,909,708	B1		6/2005	Krishnaswamy et al.	
6,925,595	B1	*	8/2005	Whitledge et al.	715/234
6,938,073	B1	*	8/2005	Mendhekar et al.	709/217
6,970,602	B1	*	11/2005	Smith et al.	382/232
7,016,963	B1	*	3/2006	Judd et al.	709/228
7,089,330	B1	*	8/2006	Mason	709/246
7,284,201	B2		10/2007	Cohen-solal	
7,313,361	B2		12/2007	Steelberg et al.	
7,406,434	B1		7/2008	Chang et al.	
7,477,688	B1		1/2009	Zhang et al.	
7,574,486	B1	*	8/2009	Cheng et al.	709/219
7,574,653	B2	*	8/2009	Croney et al.	715/249
7,673,063	B2		3/2010	Xie et al.	
7,843,437	B1	*	11/2010	Arnold et al.	345/169
2002/0103934	A1	*	8/2002	Fishman et al.	709/246
2002/0103935	A1	*	8/2002	Fishman et al.	709/246
2003/0225568	A1		12/2003	Salmonsen	
2004/0025176	A1		2/2004	Franklin et al.	
2005/0091311	A1		4/2005	Lund et al.	
2005/0190872	A1		9/2005	Seong et al.	
2005/0255852	A1		11/2005	Steelberg et al.	
2005/0278794	A1		12/2005	Leinonen et al.	
2006/0015580	A1		1/2006	Gabriel et al.	
2006/0127059	A1		6/2006	Fanning	
2007/0061198	A1		3/2007	Ramer et al.	
2007/0234213	A1		10/2007	Krikorian et al.	
2008/0155230	A1		6/2008	Robbins et al.	
2008/0186377	A1		8/2008	Eriksson et al.	
2008/0187279	A1		8/2008	Gilley et al.	
2008/0195938	A1		8/2008	Tischer et al.	
2008/0205389	A1		8/2008	Fang et al.	
2008/0207182	A1		8/2008	Maharajh et al.	
2008/0307454	A1		12/2008	Ahanger et al.	
2009/0003432	A1		1/2009	Liu et al.	
2009/0013347	A1		1/2009	Ahanger et al.	
2009/0070485	A1		3/2009	Barger et al.	
2009/0089422	A1		4/2009	Barger et al.	
2009/0240569	A1		9/2009	Ramer et al.	
2009/0254672	A1		10/2009	Zhang	
2010/0046842	A1		2/2010	Conwell	
2010/0153495	A1		6/2010	Barger et al.	
2011/0221745	A1		9/2011	Goldman et al.	
2011/0279638	A1		11/2011	Periyannan et al.	
2012/0016858	A1		1/2012	Rathod	
2012/0215629	A1		8/2012	Girouard et al.	

FOREIGN PATENT DOCUMENTS

EP	0782085	A1	7/1997
EP	0818907	A2	1/1998
EP	0843276		5/1998
EP	0876034		11/1998
EP	0883068	A2	12/1998
EP	0886409	A2	12/1998
EP	0895171	A2	2/1999
EP	0926607	A2	6/1999
EP	0949571	A2	10/1999
WO	WO-9749252		12/1997
WO	WO-98/40842		9/1998
WO	WO-9843177		10/1998

OTHER PUBLICATIONS

Berinstein, P, "The Big Picture; Text and Graphics on UMI's ProQuest Direct: The Best (Yet) of Both Words", http://www.infotoday.com/online/MarOL97/picture3.html, retrieved on Mar. 23, 2004 from website:, Mar. 1997, 11, pp.

Bulterman, D, "Models. Media and Motion: Using the Web to Support Multimedia Documents", Proceedings of 1997 Infl.Conf. on Multimedia Modeling, Singapore, Nov. 1997, 17-20, pp.

Dobson, R, "Animating Your Web Pages with Direct Animation", Web Techniques; Jun. 1998, 49-52, 5pp.

McNeil, S, "Research Interests", httpllwww.coe.uh.edul'smcneil/research.htm, Mar. 18, 2004, 3, pp.

Mohler, J, "Migrating Course Materials to the World Wide Web: A Case Study of the Department of Technical Graphics at Purdue University", Computer Networks and ISDN Systems; vol. 30, Issues 20-21,, Nov. 12, 1988, 2 pp.

Sakaguchi, et al., "A browsing tool for multi-lingual documents for users without multilingual fonts", ACM International Conference on Digital Libraries, 1996, 63-71, pp.

Zaiane, et al., "Mining multimedia data", ACM Conference of the Center for Advanced Studies on Collaborative research, Nov. 1998, 1-18, pp.

* cited by examiner

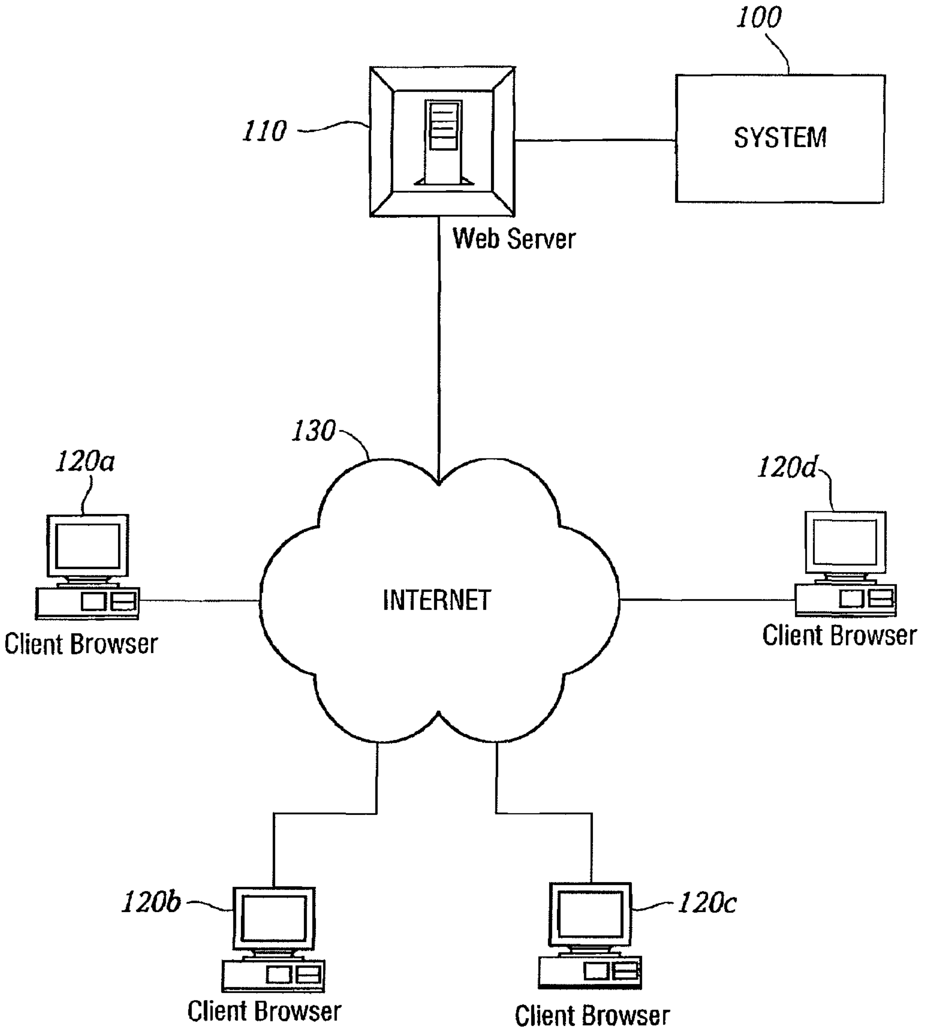

100
SYSTEM
110
Web Server
130
INTERNET
120a
Client Browser
120d
Client Browser
120b
Client Browser
120c
Client Browser

FIG. 1

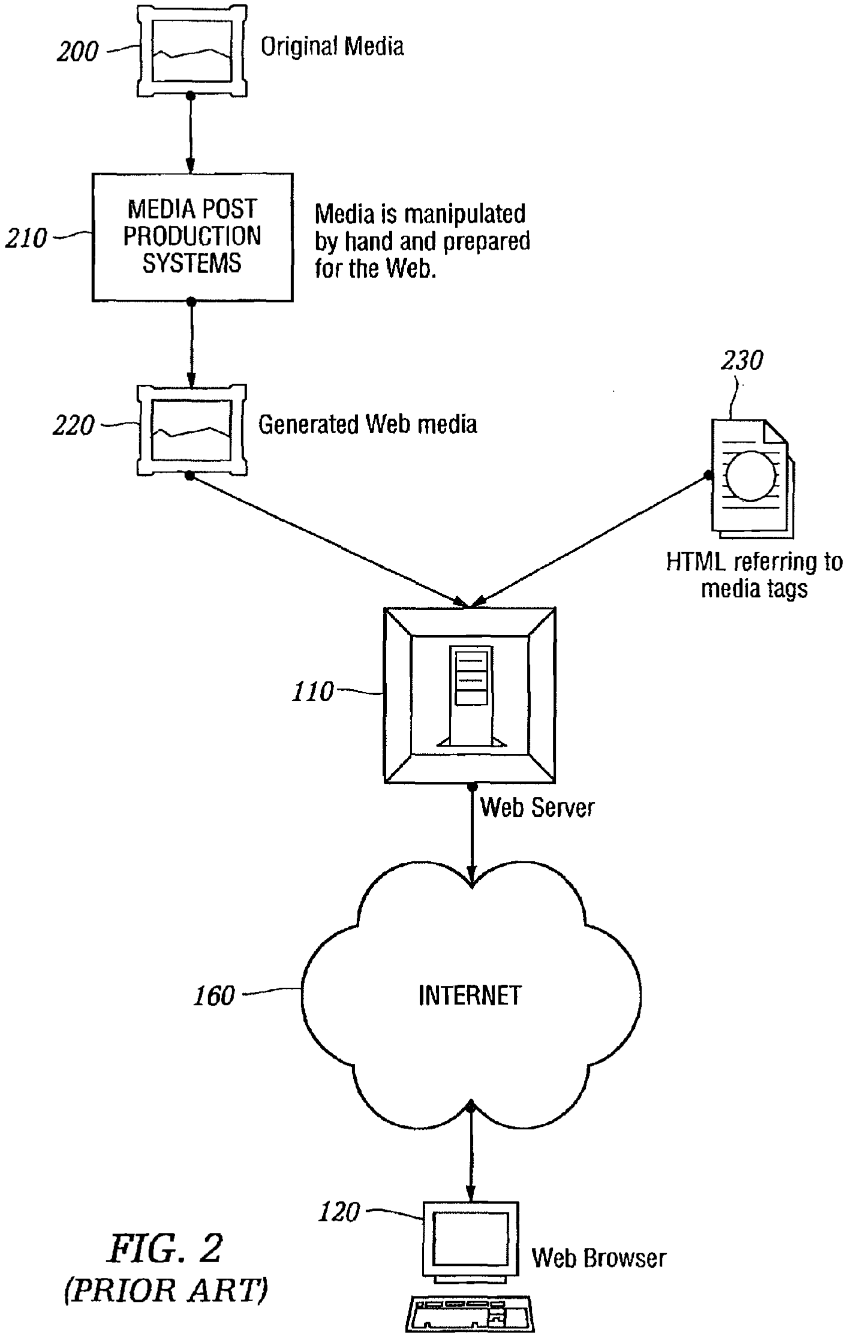

200
Original Media
210
MEDIA POST PRODUCTION SYSTEMS
Media is manipulated by hand and prepared for the Web.
220
Generated Web media
230
HTML referring to media tags
110
Web Server
160
INTERNET
120
Web Browser

FIG. 2
(PRIOR ART)

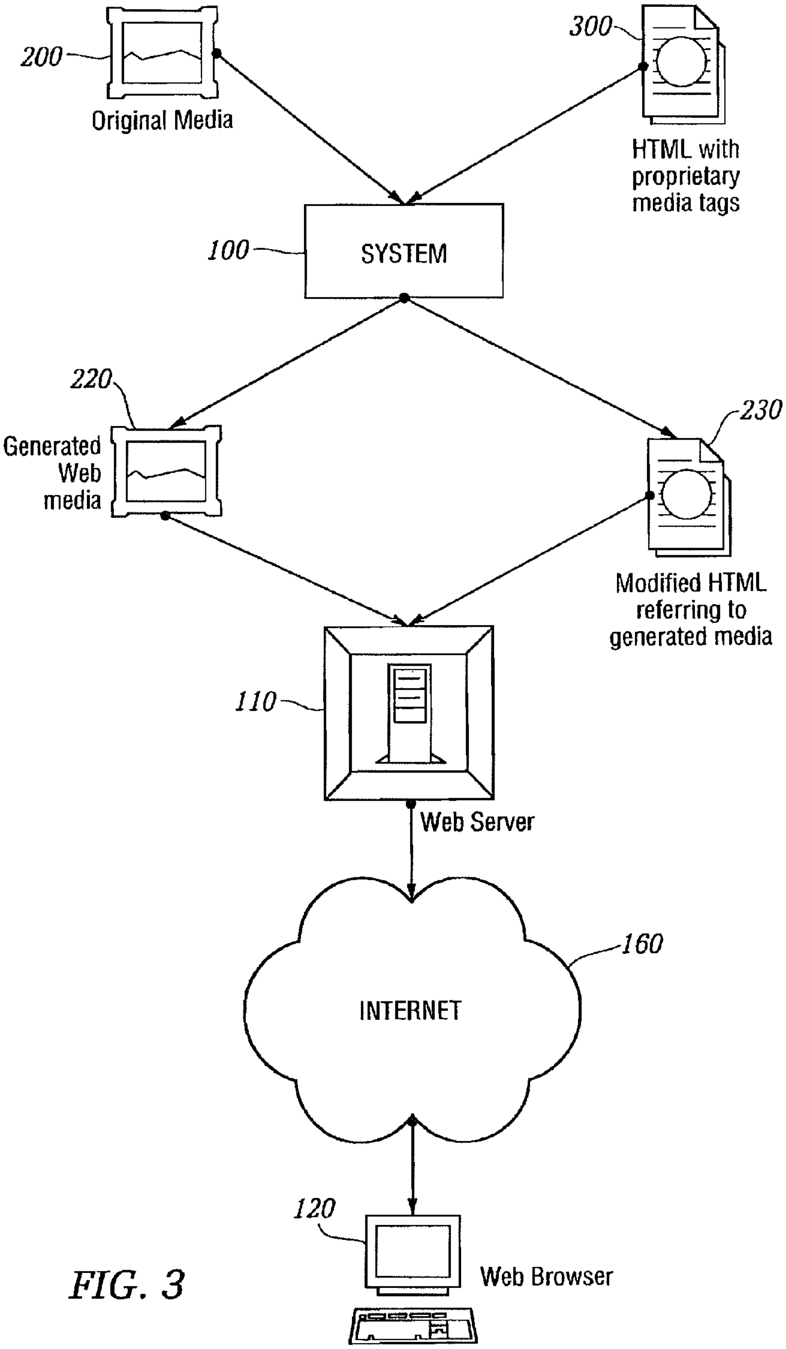

200
Original Media
300
HTML with proprietary media tags
100
SYSTEM
220
Generated Web media
230
Modified HTML referring to generated media
110
Web Server
160
INTERNET
120
Web Browser

FIG. 3

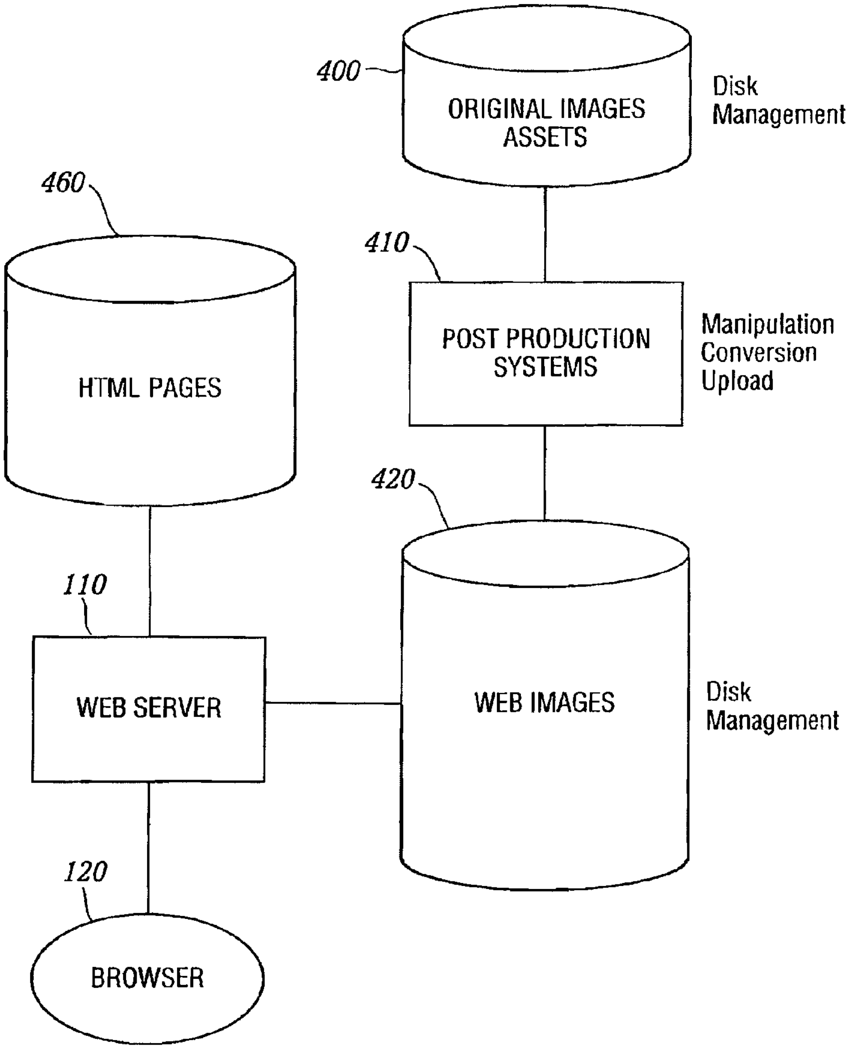


FIG. 4
(PRIOR ART)

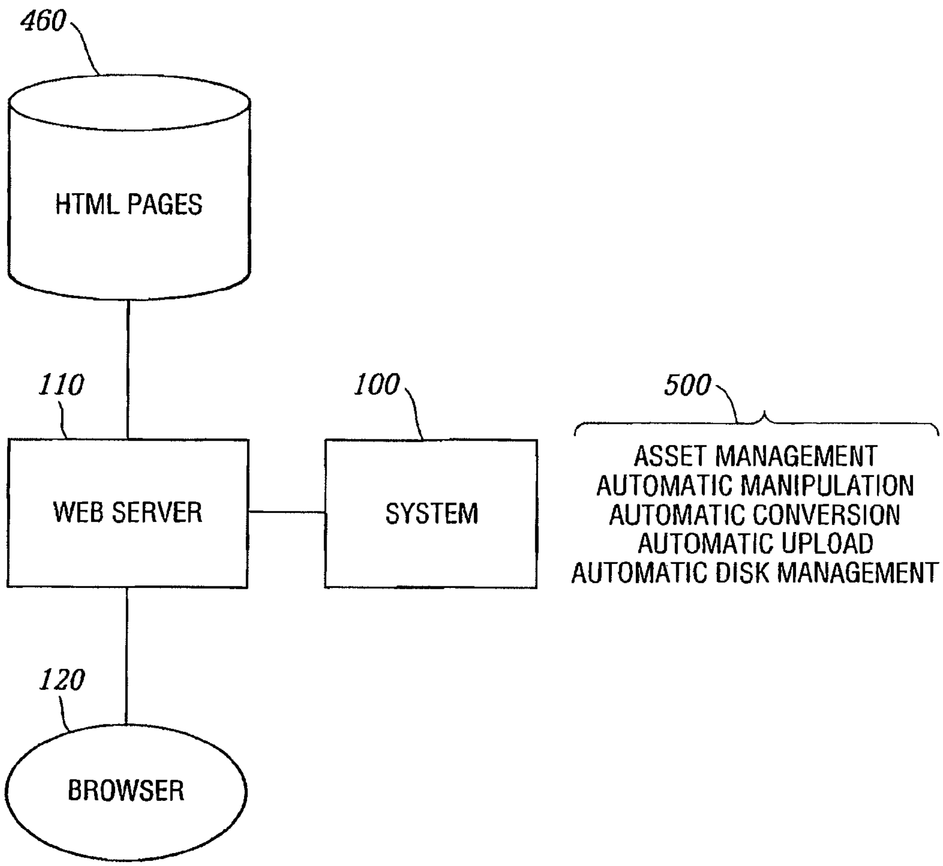

460
HTML PAGES
110
WEB SERVER
100
SYSTEM
500
ASSET MANAGEMENT
AUTOMATIC MANIPULATION
AUTOMATIC CONVERSION
AUTOMATIC UPLOAD
AUTOMATIC DISK MANAGEMENT
120
BROWSER

FIG. 5

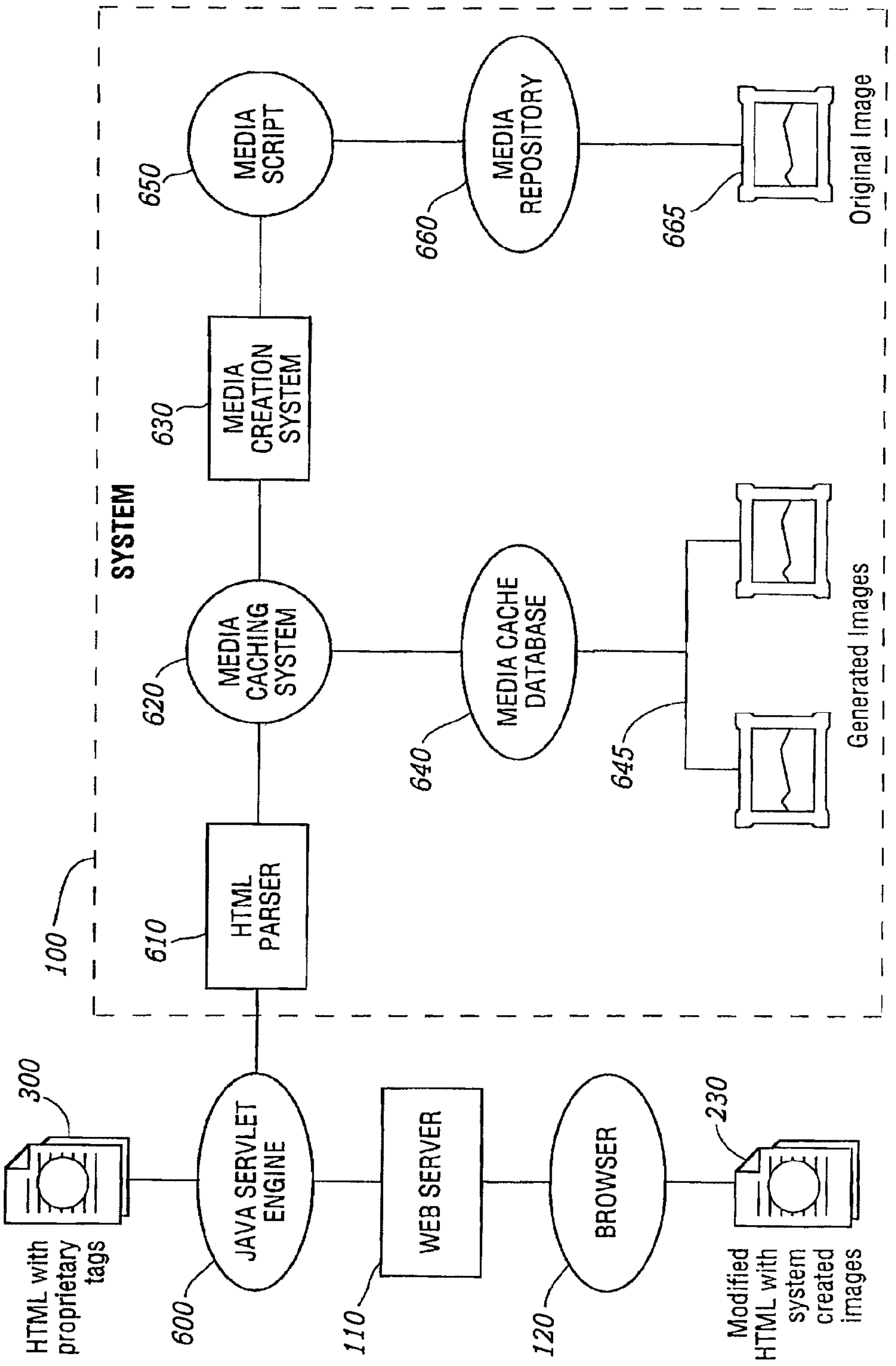

SYSTEM
100
HTML PARSER
610
MEDIA CACHING SYSTEM
620
MEDIA CREATION SYSTEM
630
MEDIA SCRIPT
650
MEDIA CACHE DATABASE
640
MEDIA REPOSITORY
660
645
Generated Images
665
Original Image
HTML with proprietary tags
300
JAVA SERVLET ENGINE
600
WEB SERVER
110
BROWSER
120
Modified HTML with system created images
230

FIG. 6

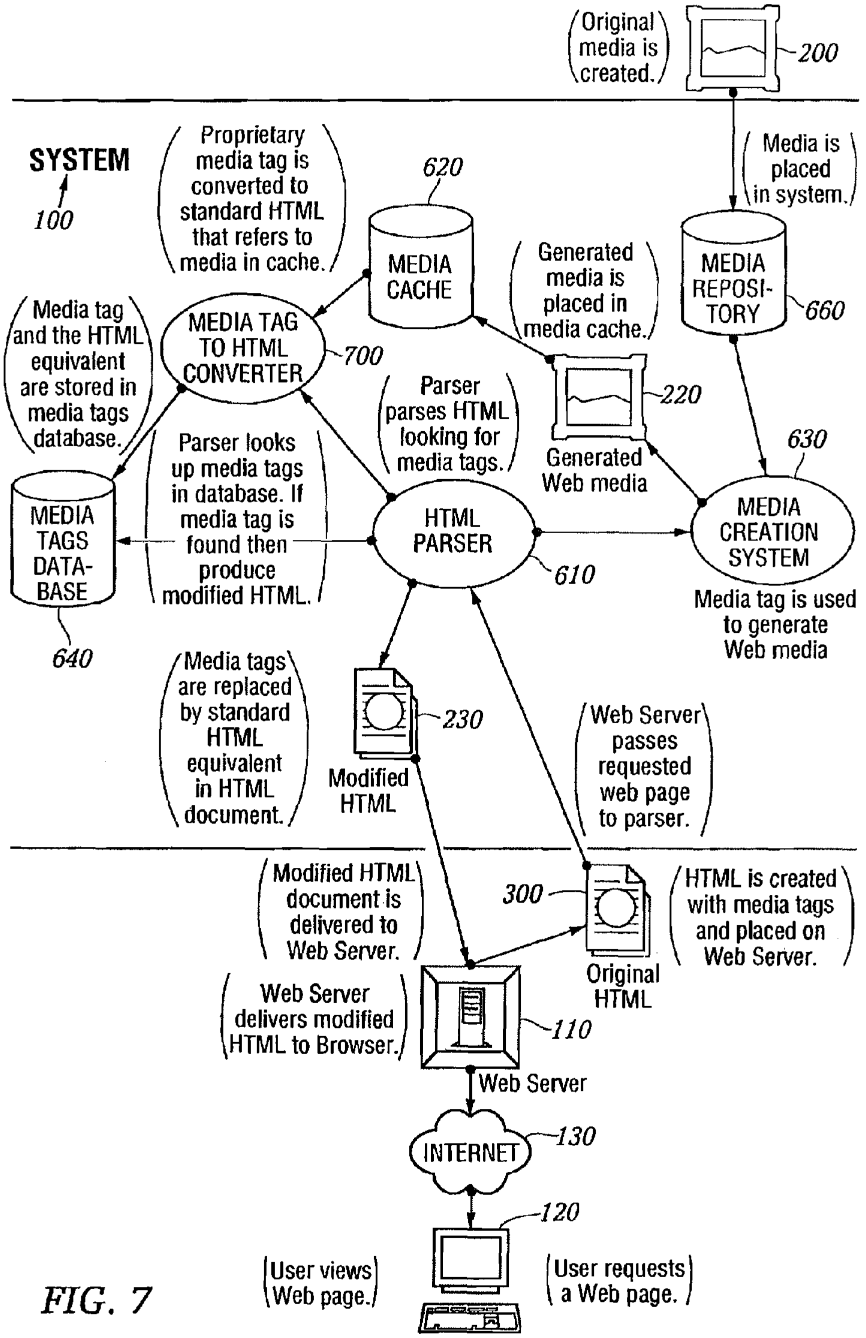

Original media is created.
200
SYSTEM
100
Proprietary media tag is converted to standard HTML that refers to media in cache.
620
MEDIA CACHE
Media is placed in system.
MEDIA REPOSITORY
660
Generated media is placed in media cache.
Media tag and the HTML equivalent are stored in media tags database.
MEDIA TAG TO HTML CONVERTER
700
Parser parses HTML looking for media tags.
220
Generated Web media
630
Parser looks up media tags in database. If media tag is found then produce modified HTML.
MEDIA TAGS DATA-BASE
640
HTML PARSER
610
MEDIA CREATION SYSTEM
Media tag is used to generate Web media
Media tags are replaced by standard HTML equivalent in HTML document.
230
Modified HTML
Web Server passes requested web page to parser.
Modified HTML document is delivered to Web Server.
300
HTML is created with media tags and placed on Web Server.
Original HTML
Web Server delivers modified HTML to Browser.
110
Web Server
INTERNET
130
120
User views Web page.
User requests a Web page.

FIG. 7

AUTHORING FLOWCHART

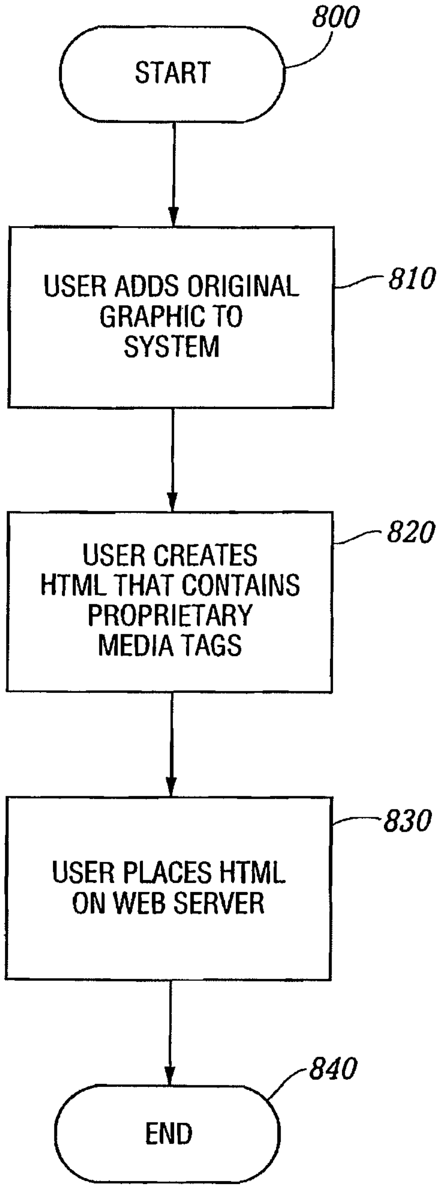

800
START
USER ADDS ORIGINAL GRAPHIC TO SYSTEM
810
USER CREATES HTML THAT CONTAINS PROPRIETARY MEDIA TAGS
820
USER PLACES HTML ON WEB SERVER
830
END
840

FIG. 8

HTML PARSING FLOWCHART

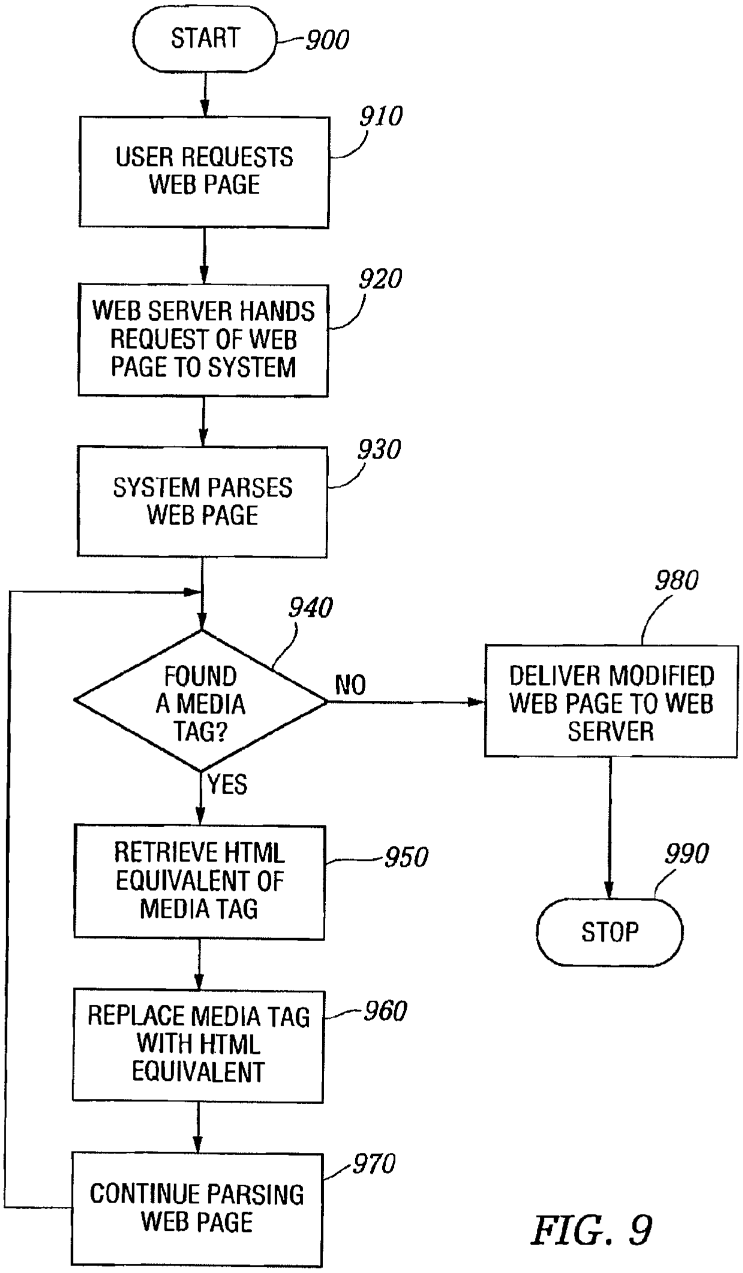

START
900
USER REQUESTS WEB PAGE
910
WEB SERVER HANDS REQUEST OF WEB PAGE TO SYSTEM
920
SYSTEM PARSES WEB PAGE
930
FOUND A MEDIA TAG?
940
NO
YES
DELIVER MODIFIED WEB PAGE TO WEB SERVER
980
STOP
990
RETRIEVE HTML EQUIVALENT OF MEDIA TAG
950
REPLACE MEDIA TAG WITH HTML EQUIVALENT
960
CONTINUE PARSING WEB PAGE
970

FIG. 9

MEDIA CREATION FLOWCHART

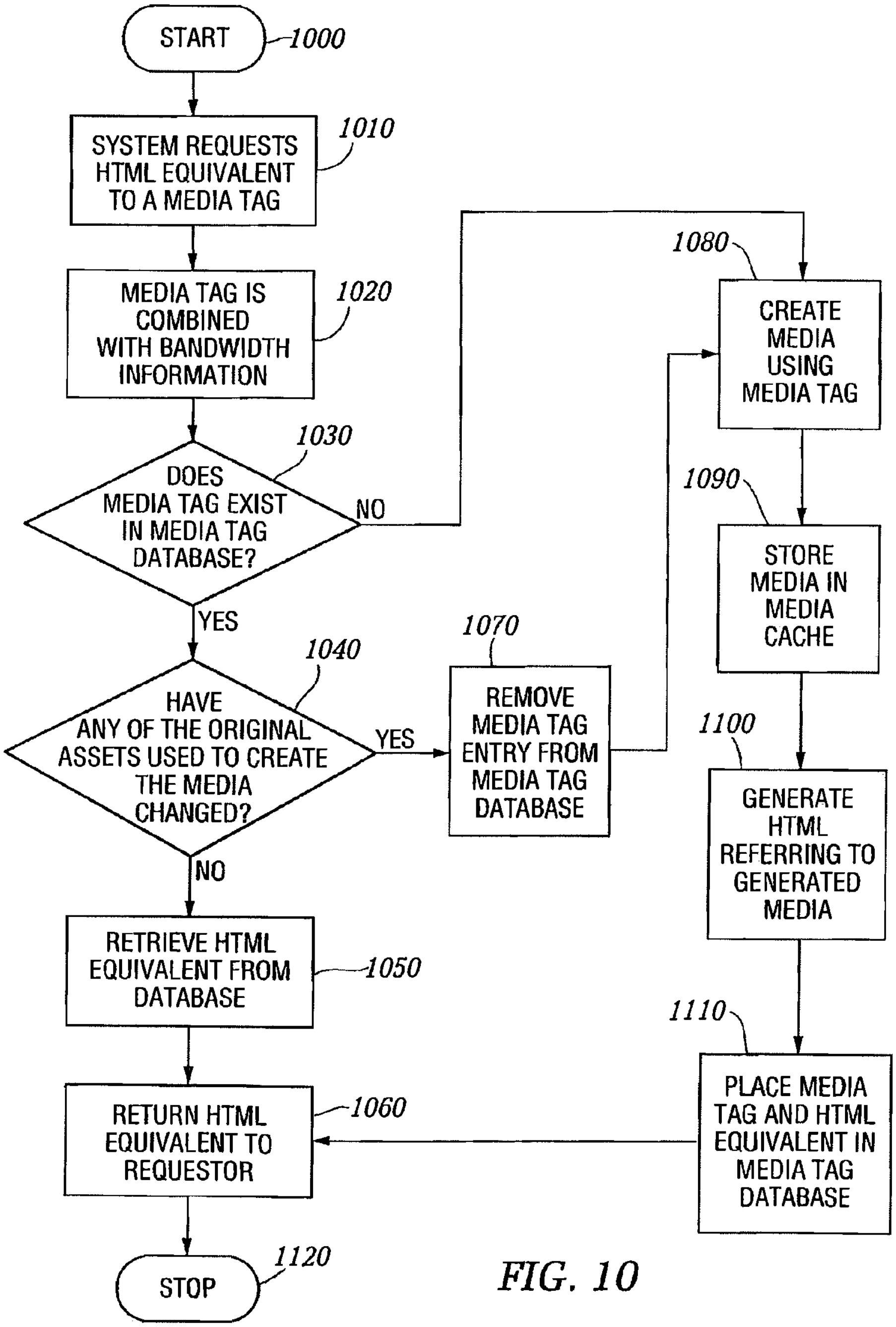
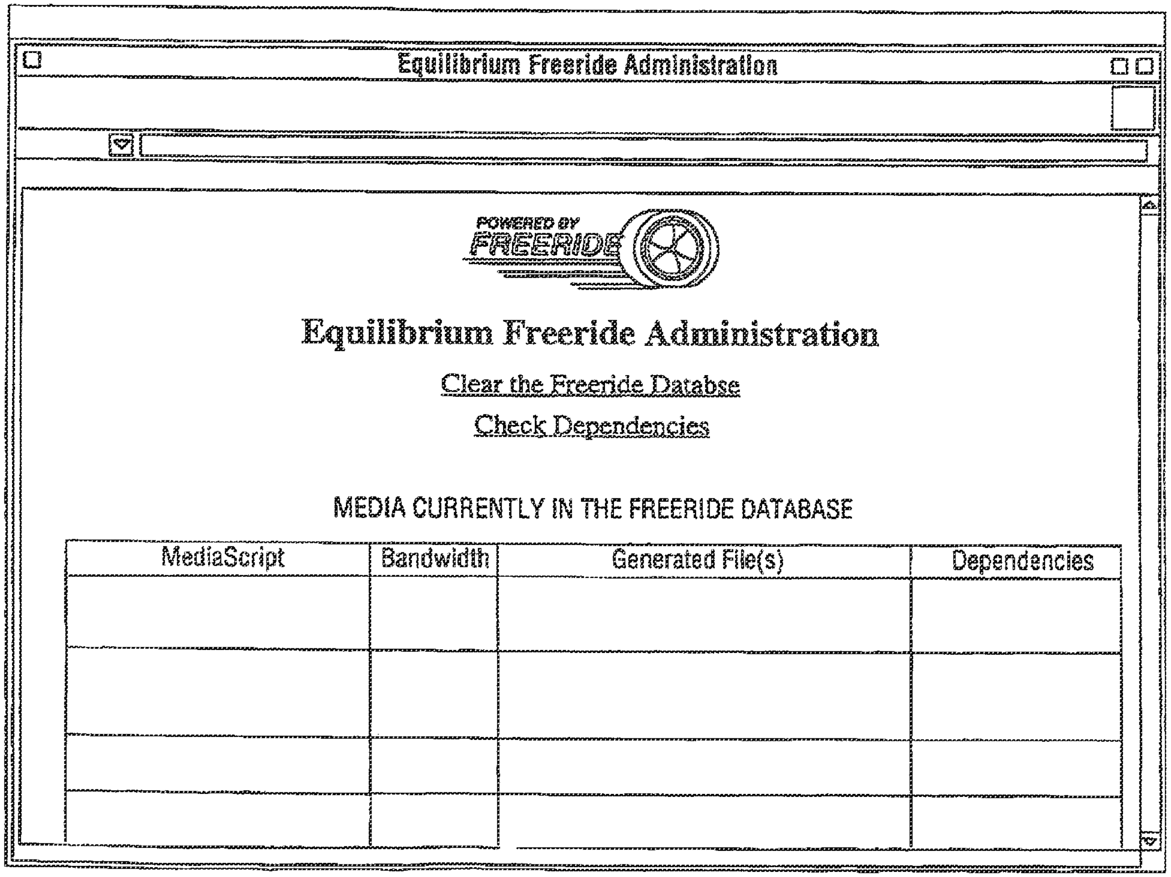

START
1000
SYSTEM REQUESTS HTML EQUIVALENT TO A MEDIA TAG
1010
MEDIA TAG IS COMBINED WITH BANDWIDTH INFORMATION
1020
DOES MEDIA TAG EXIST IN MEDIA TAG DATABASE?
1030
NO
YES
HAVE ANY OF THE ORIGINAL ASSETS USED TO CREATE THE MEDIA CHANGED?
1040
YES
NO
RETRIEVE HTML EQUIVALENT FROM DATABASE
1050
RETURN HTML EQUIVALENT TO REQUESTOR
1060
STOP
1120
REMOVE MEDIA TAG ENTRY FROM MEDIA TAG DATABASE
1070
CREATE MEDIA USING MEDIA TAG
1080
STORE MEDIA IN MEDIA CACHE
1090
GENERATE HTML REFERRING TO GENERATED MEDIA
1100
PLACE MEDIA TAG AND HTML EQUIVALENT IN MEDIA TAG DATABASE
1110

FIG. 10



Equilibrium Freeride Administration

Clear the Freeride Databse

Check Dependencies

MEDIA CURRENTLY IN THE FREERIDE DATABASE

MediaScript	Bandwidth	Generated File(s)	Dependencies

FIG. 11

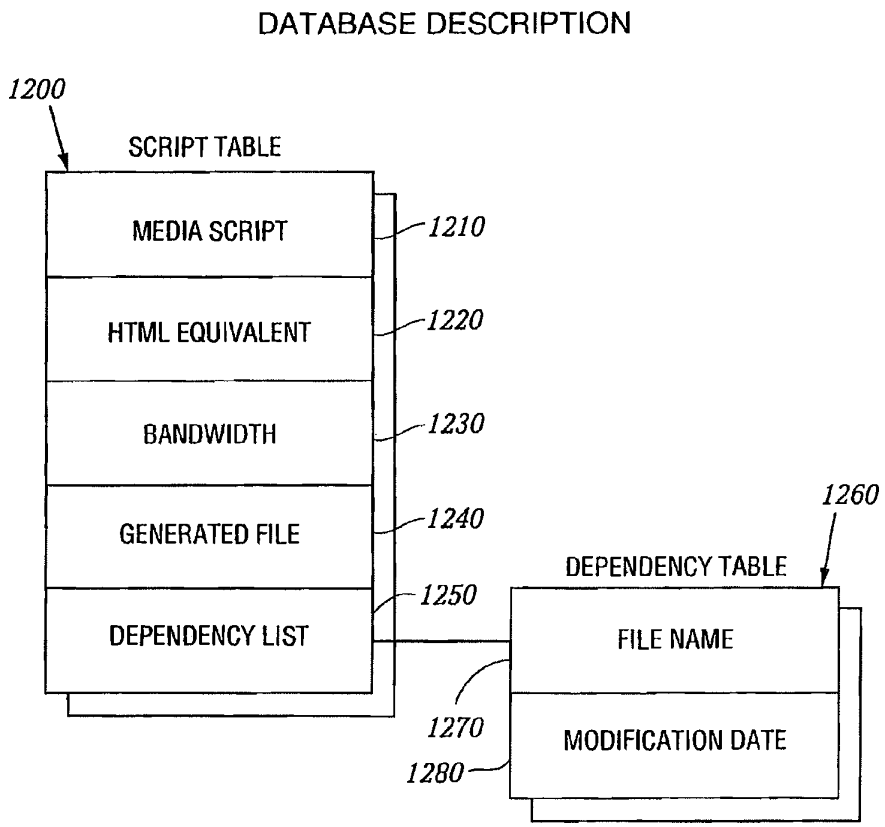

DATABASE DESCRIPTION
1200
SCRIPT TABLE
MEDIA SCRIPT
1210
HTML EQUIVALENT
1220
BANDWIDTH
1230
GENERATED FILE
1240
DEPENDENCY LIST
1250
1260
DEPENDENCY TABLE
FILE NAME
1270
MODIFICATION DATE
1280

FIG. 12

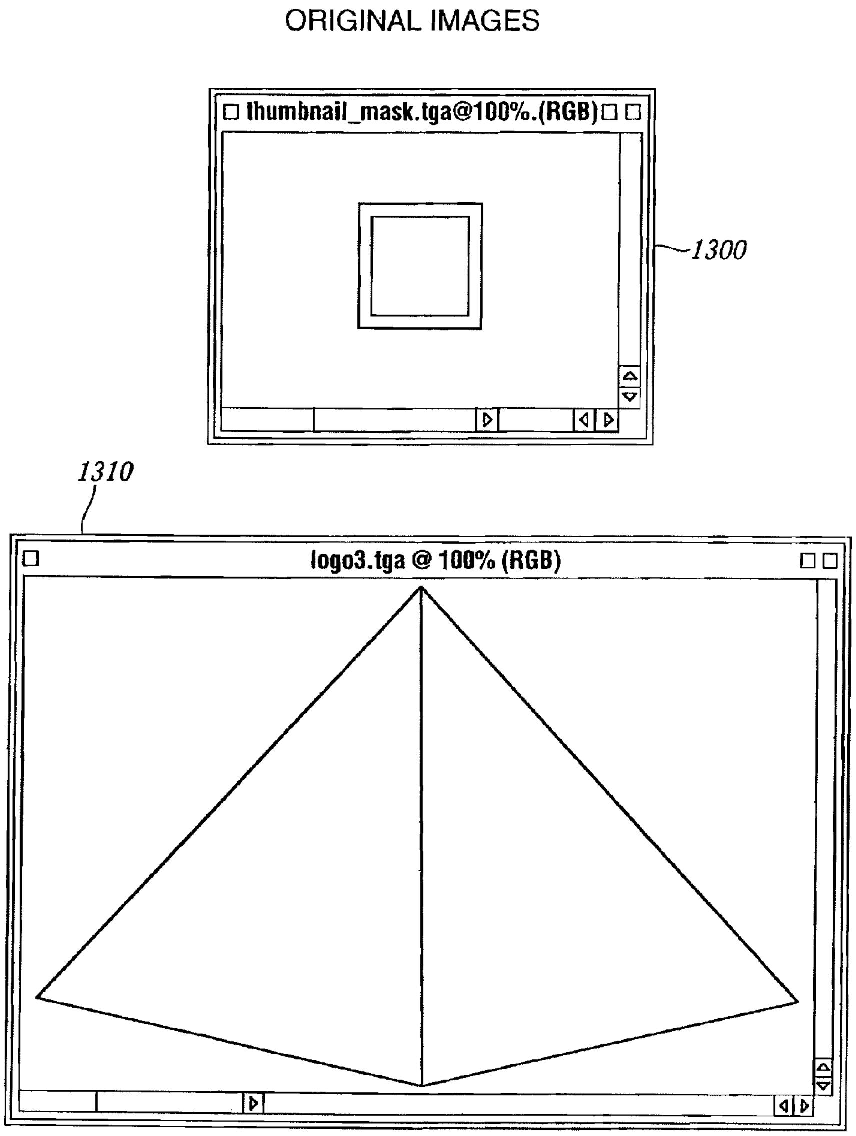
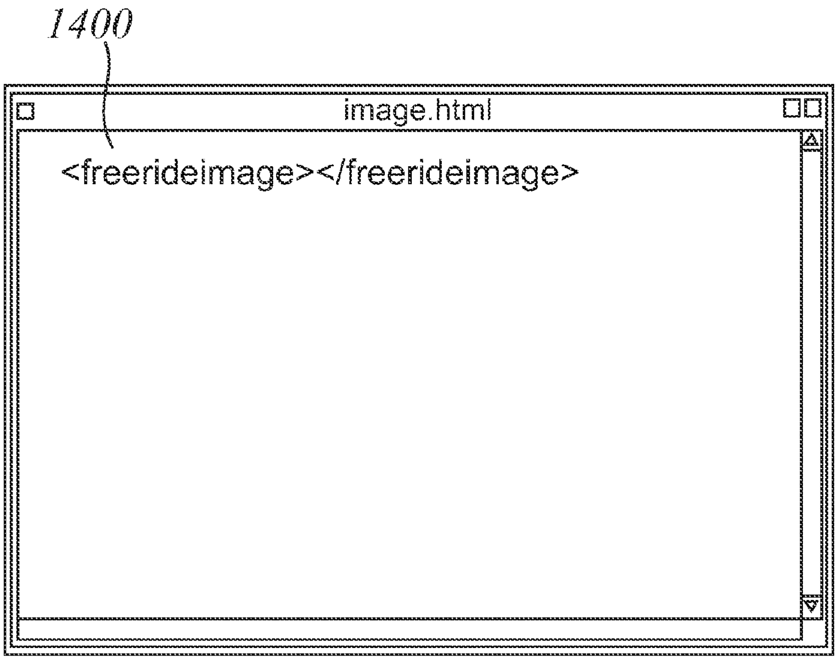

ORIGINAL IMAGES
thumbnail_mask.tga@100%.(RGB)
1300
1310
logo3.tga @ 100% (RGB)

FIG.13

HTML DOCUMENT WITH PROPRIETARY TAG



```
<freerideimage></freerideimage>
```

FIG. 14

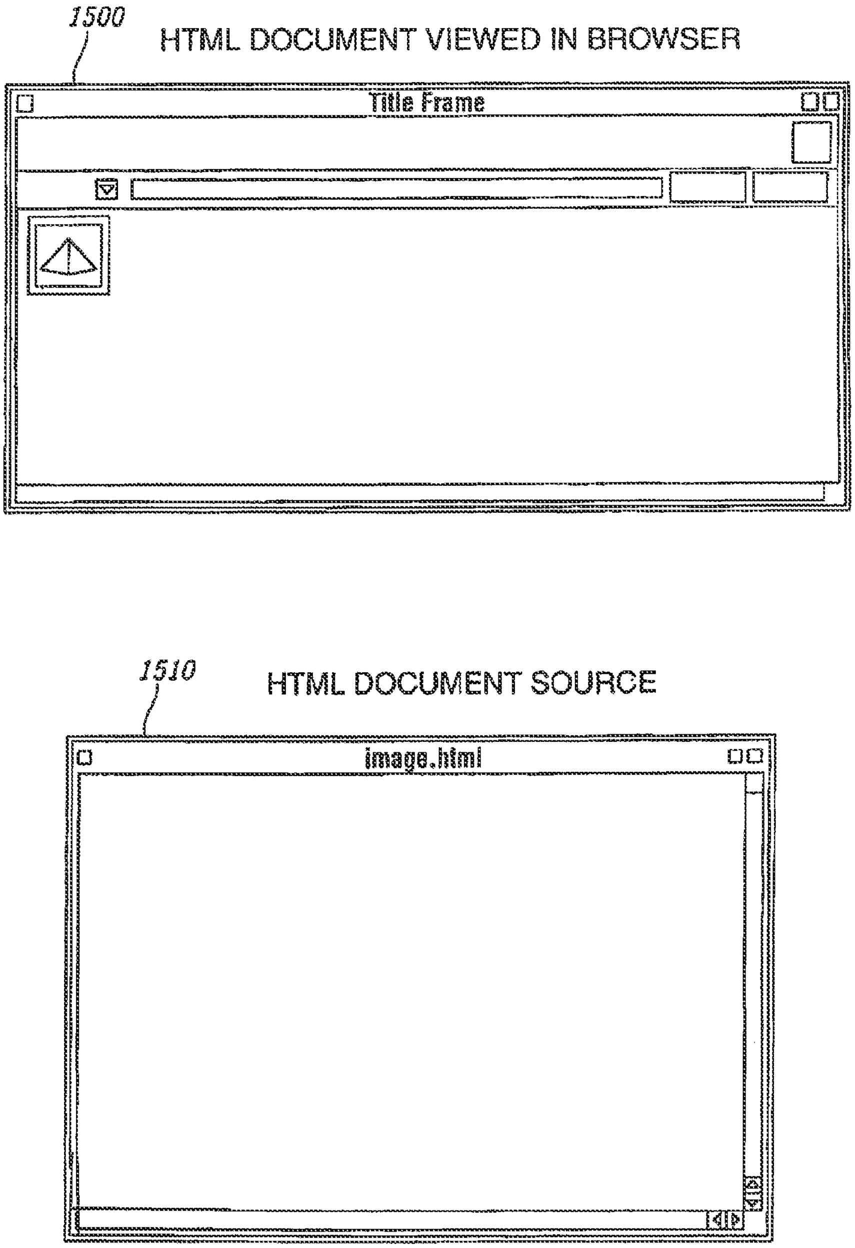

1500
HTML DOCUMENT VIEWED IN BROWSER
Title Frame
1510
HTML DOCUMENT SOURCE
image.html

FIG.15

GENERATED GIF IMAGE

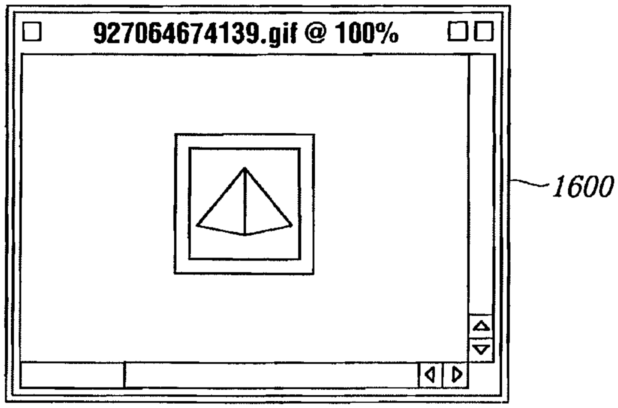


FIG.16

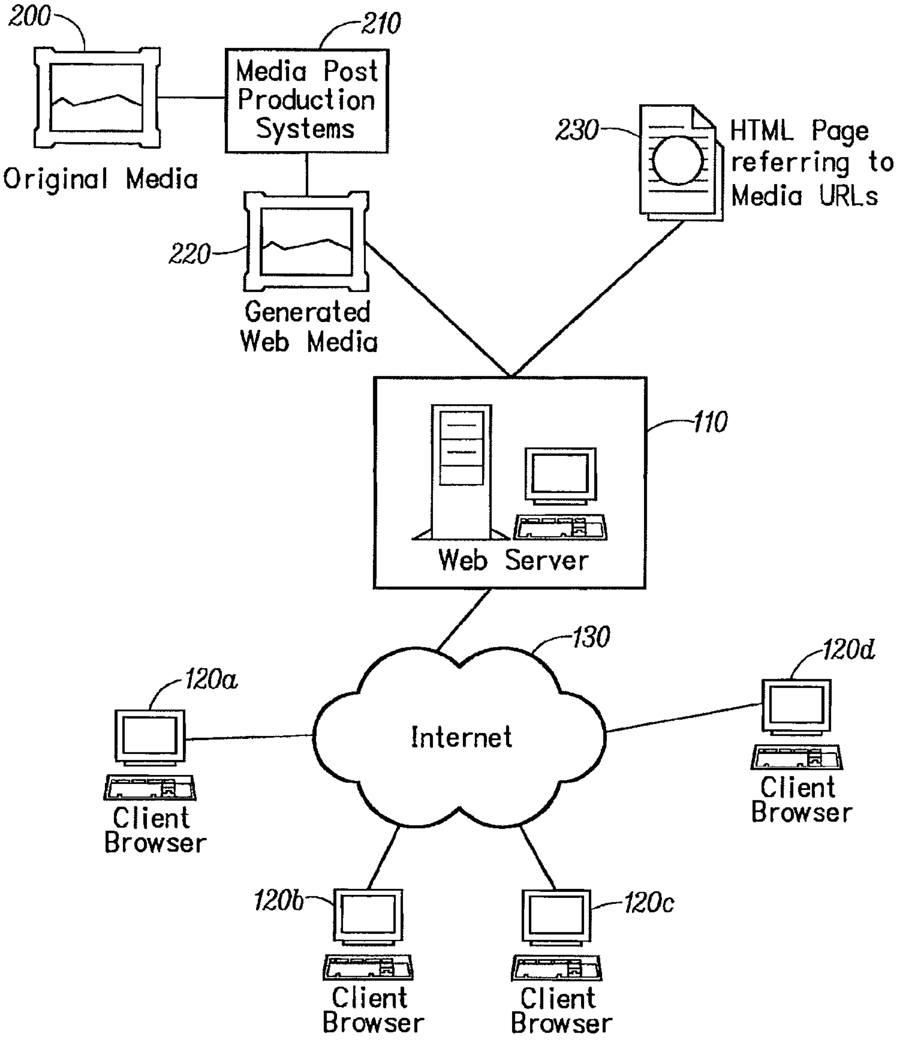

200
Original Media
210
Media Post Production Systems
230
HTML Page referring to Media URLs
220
Generated Web Media
110
Web Server
130
Internet
120a
Client Browser
120d
Client Browser
120b
Client Browser
120c
Client Browser

FIG. 17

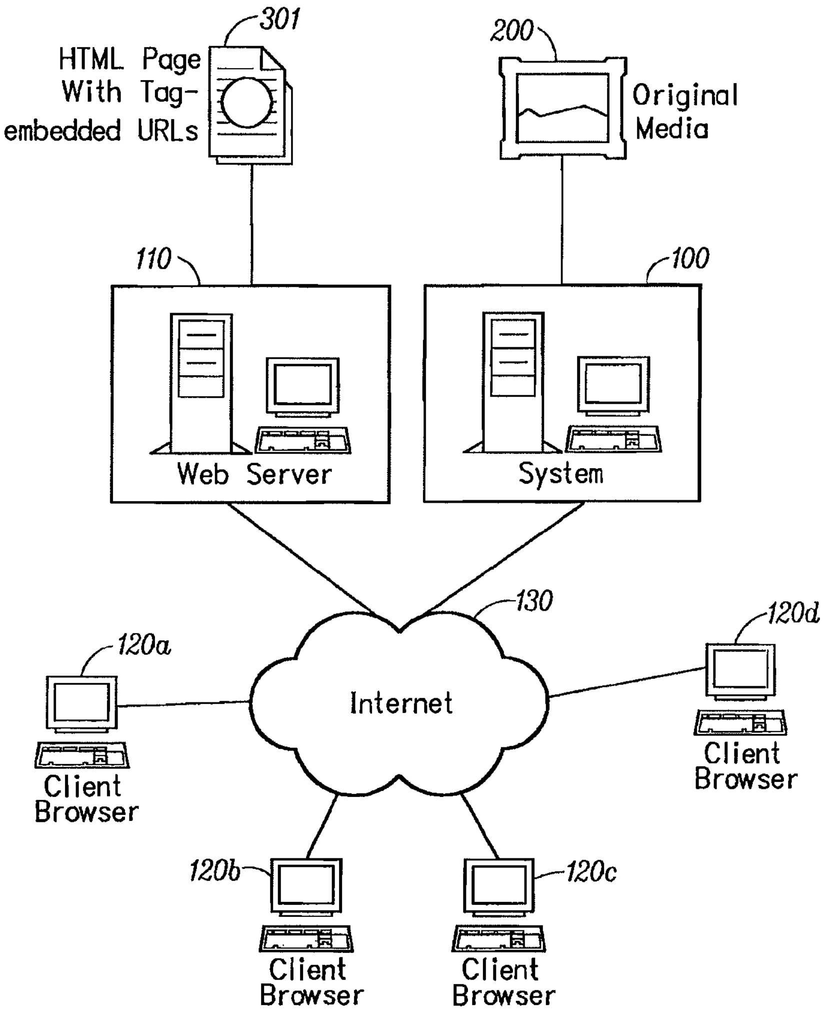

301
HTML Page With Tag-embedded URLs
200
Original Media
110
Web Server
100
System
130
Internet
120a
Client Browser
120d
Client Browser
120b
Client Browser
120c
Client Browser

FIG. 18

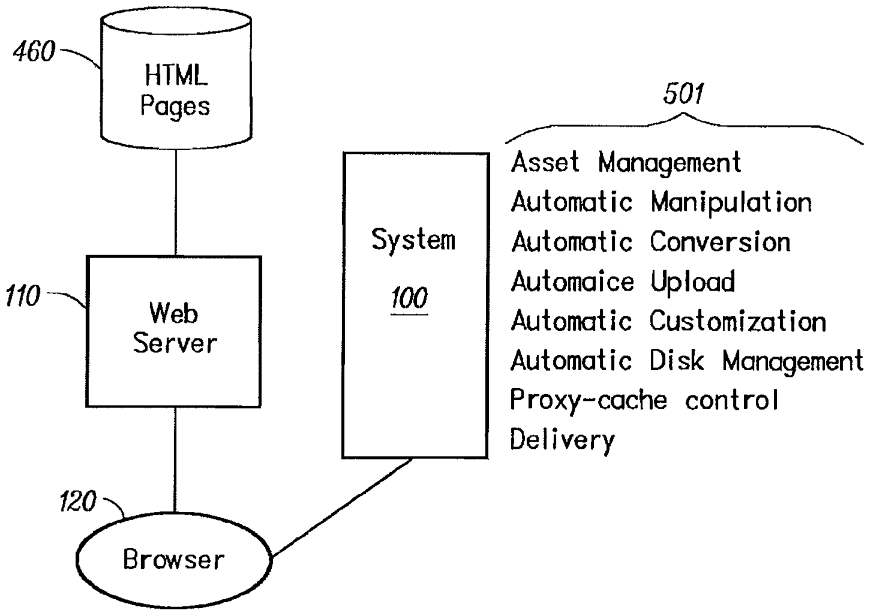

460
HTML Pages
110
Web Server
120
Browser
System
100
501
Asset Management
Automatic Manipulation
Automatic Conversion
Automaice Upload
Automatic Customization
Automatic Disk Management
Proxy-cache control
Delivery

FIG. 19

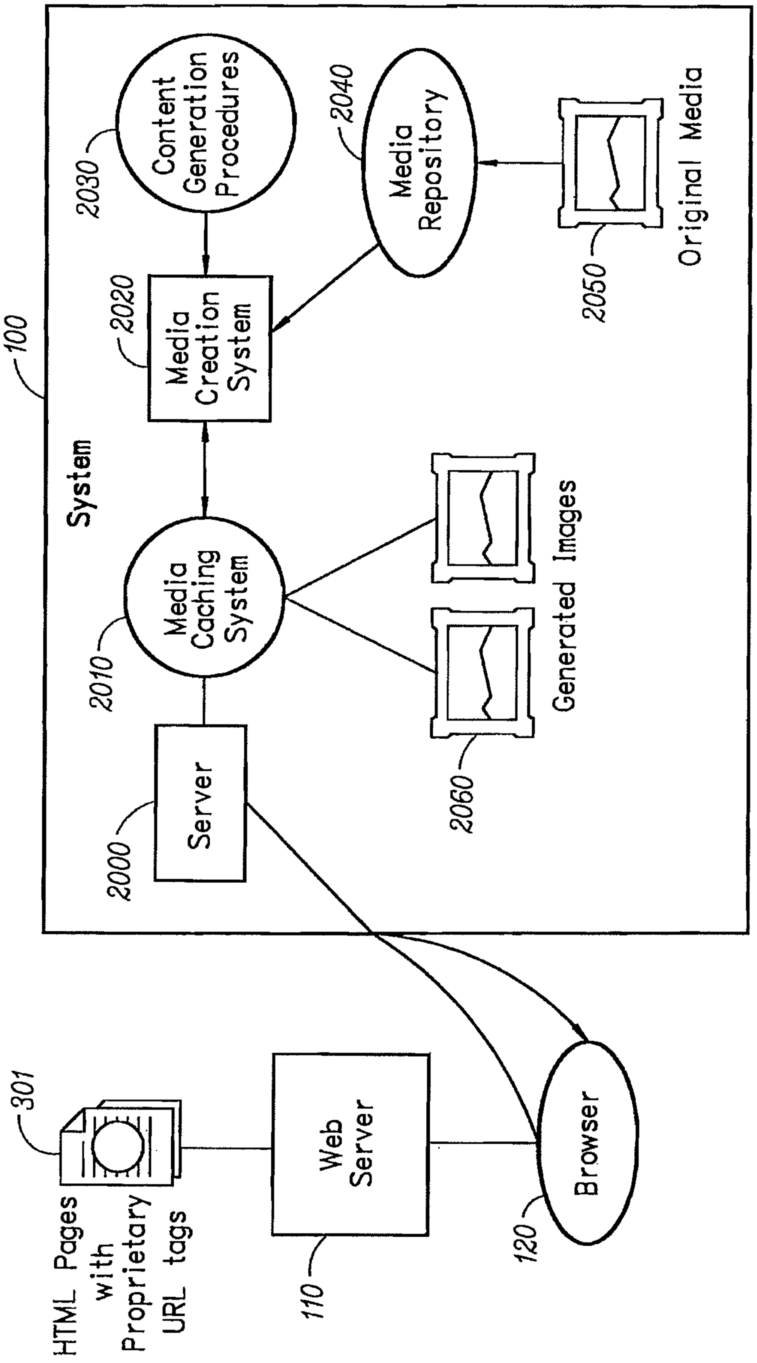

100
System
2000
Server
2010
Media Caching System
2020
Media Creation System
2030
Content Generation Procedures
2040
Media Repository
2050
Original Media
2060
Generated Images
301
HTML Pages with Proprietary URL tags
110
Web Server
120
Browser

FIG. 20

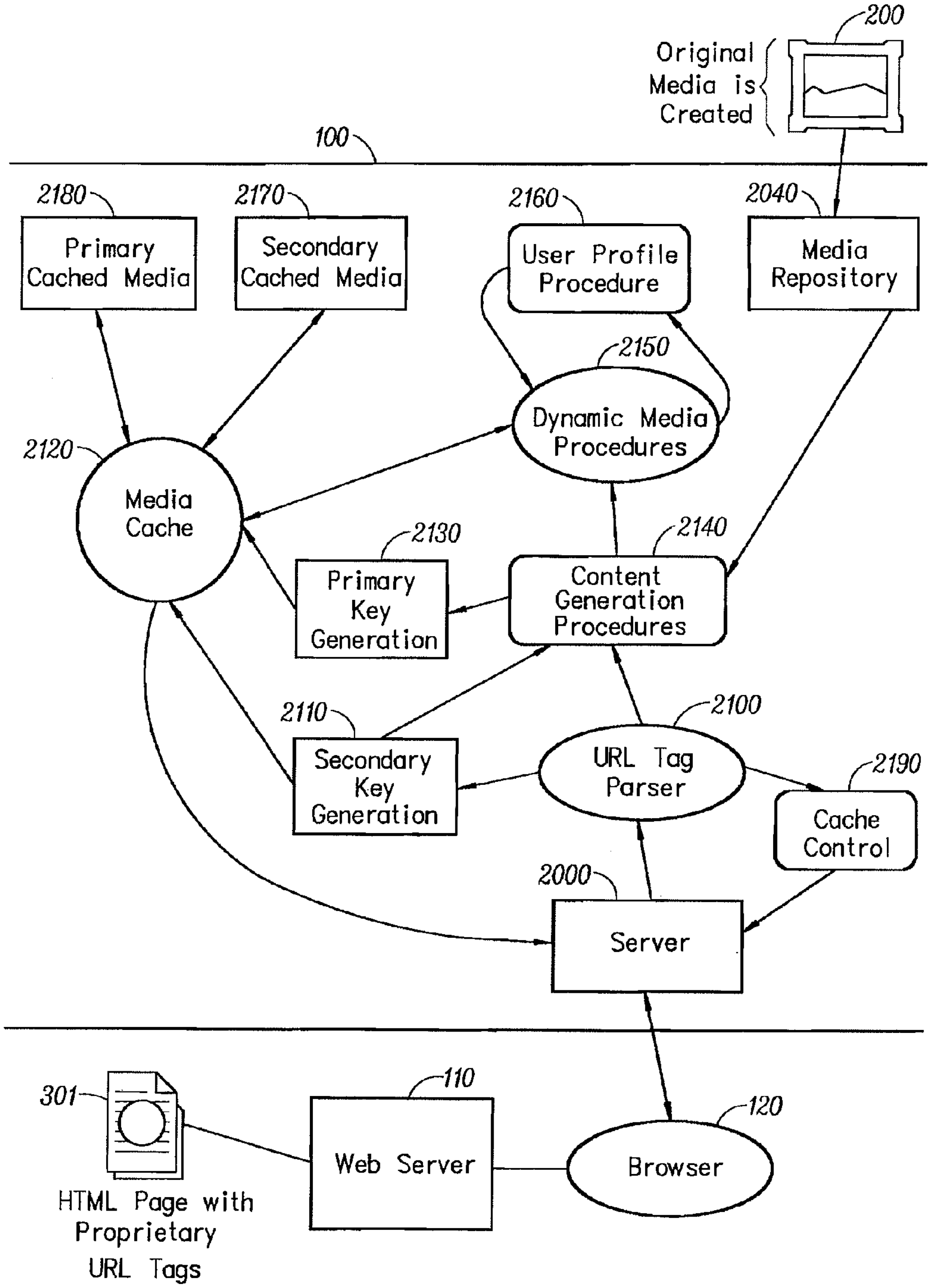

200
Original Media is Created
100
2180
Primary Cached Media
2170
Secondary Cached Media
2160
User Profile Procedure
2040
Media Repository
2150
Dynamic Media Procedures
2120
Media Cache
2130
Primary Key Generation
2140
Content Generation Procedures
2110
Secondary Key Generation
2100
URL Tag Parser
2190
Cache Control
2000
Server
301
HTML Page with Proprietary URL Tags
110
Web Server
120
Browser

FIG. 21

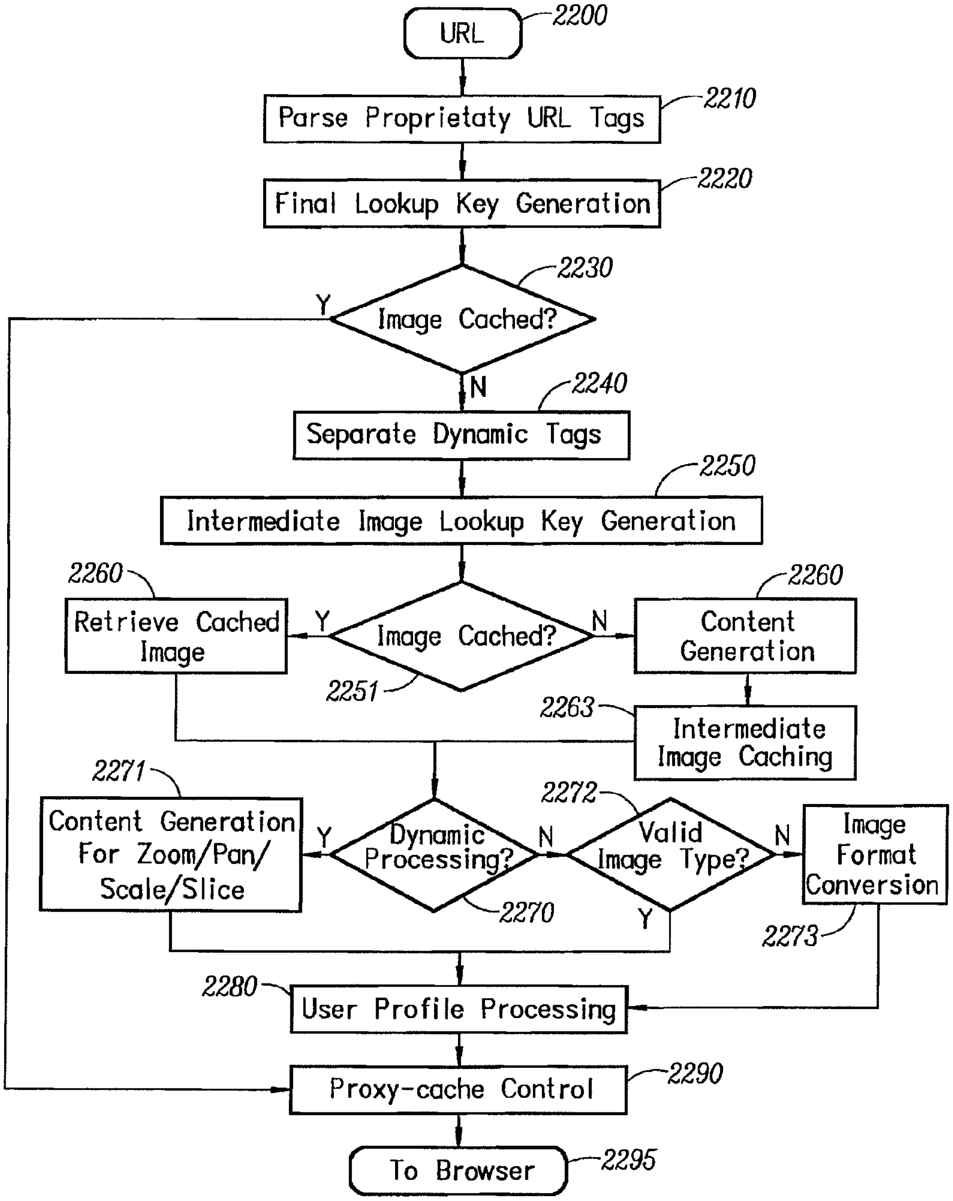

URL
2200
Parse Proprietaty URL Tags
2210
Final Lookup Key Generation
2220
Image Cached?
2230
Y
N
Separate Dynamic Tags
2240
Intermediate Image Lookup Key Generation
2250
2260
Retrieve Cached Image
Y
Image Cached?
N
2251
2260
Content Generation
2263
Intermediate Image Caching
2271
Content Generation For Zoom/Pan/ Scale/Slice
Y
Dynamic Processing?
2270
N
2272
Valid Image Type?
N
Y
Image Format Conversion
2273
2280
User Profile Processing
Proxy-cache Control
2290
To Browser
2295

FIG. 22

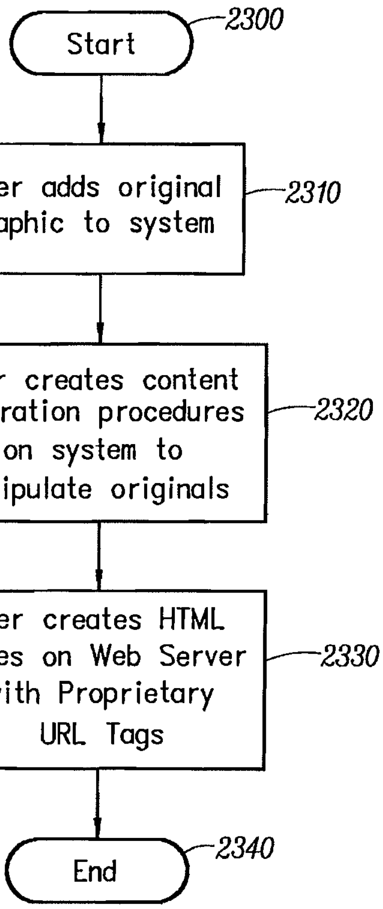

Start
2300
User adds original graphic to system
2310
User creates content generation procedures on system to manipulate originals
2320
User creates HTML pages on Web Server with Proprietary URL Tags
2330
End
2340

FIG. 23

OPTIMIZATION OF MEDIA CONTENT USING GENERATED INTERMEDIATE MEDIA CONTENT

CROSS REFERENCE TO RELATED APPLICATIONS

This application is a Continuation of U.S. Ser. No. 12/238,842, filed Sep. 26, 2008, which is a Divisional of U.S. Ser. No. 12/173,747, filed Jul. 15, 2008, which is a Divisional of U.S. Ser. No. 11/269,916, filed Nov. 7, 2005, which is a Continuation-in-Part of U.S. Ser. No. 09/929,904, filed Aug. 14, 2001, now U.S. Pat. No. 6,964,009 granted on Nov. 8, 2005, which is a Continuation-in-Part of U.S. Ser. No. 09/425,326, filed Oct. 21, 1999, now U.S. Pat. No. 6,792,575, granted on Sep. 14, 2004, each of which is hereby incorporated in its entirety by this reference thereto.

BACKGROUND OF THE INVENTION

1. Technical Field

The invention relates to software systems. More particularly, the invention relates to an Internet server-based software system that provides delivery of automated graphics and other media to Web sites for access by an end user or consumer.

2. Description of the Prior Art

Most Web sites today are primarily handmade. From the guy publishing a simple online technology newsletter from his home, to the Fortune 1000 company's multi-tiered site with hundreds of pages of text, images, and animations, the Web developer and each of his HTML-coding and graphics-producing coworkers toil page by page and image by image. Thousands of established online companies employ hundreds of highly-skilled workers just to produce and maintain their Web sites. After all, the Web is now a major selling vehicle and marketing medium for many of these companies. The Web has even sprouted service industries such as, for example, public companies with multi-billion dollar valuations created just to consult and produce Web sites for others.

Most Web developers who use established WYSIWYG tools in the industry still must produce each page on their Web site one by one. The same rate applies to preparing and placing images, animations, and other visual assets. Each page represents its own set of issues ranging from whether to use GIF, JPEG, or PNG file formats, to finding the optimum bit depth for each image to ensure the fastest downloading through the different browsers of the consumer. The bottlenecked state of the customer's workflow to produce graphics for Web pages can be described as follows:

Current Workflow for Creating Web Graphics
Original Artwork/Asset Creation
  Use third-party point products
Asset Editing
  Scale/reduce/slice
Asset Format Conversion
  JPEG/GIF/PNG
Asset Staging
  Place in Web file system
  Edit HTML
Create/Modify HTML for particular page
Store HTML on Web server
View final pages
Repeat process for each version of each graphic on each page
Estimated Time
Two hours per page times the number of pages

Also, from a user's perspective, the current state of the art is to offer the consumer zooming and panning capabilities so that by clicking on an image the consumer can view more closely or from a different angle. On the horizon are pages with three-dimensional imagery that enable a user to move around a page that can look more like a room than a brochure. While interesting, these features are merely incremental improvements to a consumer's surfing experience.

D. C. A. Bulterman, *Models, Media, and Motion: Using the Web to Support Multimedia Documents*, Proceedings of 1997 International Conference on Multimedia Modeling, Singapore, 17-20 Nov. 1997 discloses "an effort underway by members of industry, research centers and user groups to define a standard document format that can be used in conjunction with time-based transport protocols over the Internet and intranets to support rich multimedia presentations. The paper outlines the goals of the W3C's Synchronized Multimedia working group and presents an initial description of the first version of the proposed multimedia document model and format."

Text and Graphics on UMI's ProQuest Direct: The Best (yet) of both Worlds, Online, vol. 21, no. 2, pp. 73-7, March-April 1997 discloses an information system that offers "periodical and newspaper content covering a wide range of business, news, and professional topics . . . letting the user search both text and graphics and build the product to suit. Articles can be retrieved in varying levels of detail: citation, abstracts, full text, and text with graphics. Images come in two flavors: Page Image, a virtual photocopy, and Text+Graphics, in which graphics are stored separately from the text and are manipulable as discrete items . . . . [The system] comes in two versions: Windows and Web."

John Mills Dudley, Network-Based Classified Information Systems, AU-A-53031/98 (27/08/98) discloses a "system for automatically creating databases containing industry, service, product and subject classification data, contact data, geographic location data (CCG-data) and links to web pages from HTML, XML, or SGML encoded web pages posted on computer networks such as Internets or Intranets . . . . The . . . databases may be searched for references (URLs) to web pages by use of enquiries which reference one or more of the items of the CCG-data. Alternatively, enquiries referencing the CCG-data in the databases may supply contact data without web page references. Data duplication and coordination is reduced by including in the web page CCG-data display controls which are used by web browsers to format for display the same data that is used to automatically update the databases."

Cordell et al, Automatic Data Display Formatting with A Networking Application, U.S. Pat. No. 5,845,084 (Dec. 1, 1998) discloses a placeholder image mechanism. "When a data request is made, the data transfer rate is monitored. When the receive data transfer rate is slow, and the data contains an embedded graphical image of unknown dimensions, a small placeholder image is automatically displayed for the user instead of the actual data. The small placeholder image holds a place on a display device for the data or the embedded graphical image until the data or embedded graphical image is received. When embedded graphical image is received, the placeholder image is removed, and the display device is reformatted to display the embedded graphical image."

Jonathon R. T. Lewis, System For Substituting Tags For Non-Editable Data Sets In Hypertext Documents And Updating Web Files Containing Links Between Data Sets Corresponding To Changes Made To The Tags, U.S. Pat. No. 5,355,472 (Oct. 11, 1994) discloses a "hypertext data processing system wherein data sets participating in the hypertext document may be edited, the data processing system inserting tags into the data sets at locations corresponding to the hypertext links to create a file which is editable by an editor and the data processing system removing the tags, generating a revised data set and updating the link information after the editing process. Its main purpose is to preserve the linking hierarchy that may get lost when the individual data sets get modified."

Wistendahl et al, System for Mapping Hot Spots in Media Content Interactive Digital Media Program, U.S. Pat. No. 5,708,845 (Jan. 13, 1998) discloses a "system for allowing media content to be used in an interactive digital media (IDM) program [that] has Frame Data for the media content and object mapping data (N Data) representing the frame addresses and display location coordinates for objects appearing in the media content. The N Data are maintained separately from the Frame Data for the media content, so that the media content can be kept intact without embedded codes and can be played back on any system. The IDM program has established linkages connecting the objects mapped by the N Data to other functions to be performed in conjunction with display of the media content. Selection of an object appearing in the media content with a pointer results in initiation of the interactive function. A broad base of existing non-interactive media content, such as movies, videos, advertising, and television programming can be converted to interactive digital media use. An authoring system for creating IDM programs has an object outlining tool and an object motion tracking tool for facilitating the generation of N Data. In a data storage disk, the Frame Data and the N Data are stored on separate sectors. In a network system, the object mapping data and IDM program are downloaded to a subscriber terminal and used in conjunction with presentation of the media content."

Rogers et al, Method for Fulfilling Requests of A Web Browser. U.S. Pat. No. 5,701,451 (Dec. 23, 1997) and Lagarde at al, Method for Distributed Task Fulfillment of Web Browser Requests, U.S. Pat. No. 5,710,918 (Jan. 20, 1998) disclose essentially "improvements which achieve a means for accepting Web client requests for information, obtaining data from one or more databases which may be located on multiple platforms at different physical locations on an Internet or on the Internet, processing that data into meaningful information, and presenting that information to the Web client in a text or graphics display at a location specified by the request."

Tyan et al, HTML Generator, European Patent Application No. EP 0843276 (May 20, 1998) discloses "generating an HTML file based on an input bitmap image, and is particularly directed to automatic generation of an HTML file, based on a scanned-in document image, with the HTML file in turn being used to generate a Web page that accurately reproduces the layout of the original input bitmap image."

TrueSpectra has a patent pending for the technology employed in its two products, IrisAccelerate and IrisTransactive. These products are designed for zooming and panning and simple image transformations and conversions, respectively. They support 10 file formats and allow developers to add new file formats via their SDK. They do not require the use of Flashpix for images. However, their documentation points out that performance is dependent on the Flashpix format. The system would be very slow if a non-Flashpix format was used.

TrueSpectra allows the image quality and compression to be set for JPEGs only. The compression setting is set on the server and all images are delivered at the same setting.

TrueSpectra has a simple caching mechanism. Images in the cache can be cleared out automatically at certain times and it does not have any dependency features for image propagation. The Web server needs to be brought down in order to update any original assets.

TrueSpectra does not require plug-ins to operate features such as zooming/panning or compositing. The alternative to plug-ins is using their Javascript or active server page technology. These technologies are used by many Web sites to provide interactivity, but not all Web browsers work correctly with these technologies.

TrueSpectra relies on Flashpix as its native file format and does not support media types such as multi-GIFs and sound formats. Flashpix files are typically larger than most file formats. Access to files is faster for zooming and panning, but appears to be quite slow.

The key to IrisTransactive is the compositing subsystem. It requires three things to build a shopping solution using image composition.

1) The original images must be created. It is suggested that the image be converted to Flashpix for better performance.
2) All of the individual images must be described in XML using the image composer program. The program allows the editor to specify anchor points, layer attributes, and layer names. The resulting file is between 5 k and 50 k.
3) The Web designer must place HTML referring to the XML in the Web site. By specifying parameters to the XML, the Web designer can turn on or off layers.

The herein above process for compositing images enables Web designers to create shopping sites. However, a lot of overhead is the result. The XML documents add 5 k-50 k to a Web site. The compositing commands that are embedded in the HTML are difficult to understand. And, because the compositing feature requires several steps to implement, it is not suitable for every image on a Web site. The process seems to be designed for the specific purpose of shopping.

MediaBin™ is limited to activities behind the firewall automating only the "post-creative busywork." In addition, MediaBin requires the use of an application server to function through a web interface. Thus images may not be directly added to any existing web page.

Macromedia's Generator operates by embedding variables in their proprietary Flash format. Therefore the actual imaging operations are somewhat limited and cannot be controlled directly from a web page request.

MGI Software sells point solutions that require end-users to download a viewer to process a proprietary image format.

PictureIQ offers a server-side image-processing appliance that provides a limited set of Photoshop functionalities. This appliance runs on the web-page server, processes information embedded in the web page, and rewrites the web page with image data.

The disclosed prior art fail to provide systems and methodologies that result in a quantum leap in the speed with which they can modify and add images, video, and sound to sites, in the volume of data they can publish internally and externally, and in the quality of the output. The development of such an automated media delivery system would constitute a major technological advance.

It would be advantageous to empower an end user with flexibility and control by providing interactive page capabilities.

It would be advantageous from an end user's perspective to generate Web pages that contain active graphics. For example, clicking on a Corvette image will cause a simple menu to pop up suggesting alternative colors and sizes in which to see the car. Clicking on portions of the image, such as a fender, can call up a close-in view of the fender.

It would be advantageous to provide an automated graphics delivery system that becomes part of the Web site infrastructure and operates as part of the Web page transaction and that thereby provides a less expensive and less time-consuming process.

It would be advantageous to provide a system for automated processing and delivery of media (images, video, and sound) to a Web server whereby it eliminates the laborious post-production and conversion work that must be done before a media asset can be delivered on a Web server.

It would be advantageous to create a dynamic Web site, wherein images are generated on demand from original assets, wherein only the original assets need to be updated, and wherein updated changes propagate throughout the site.

It would be advantageous to provide a system that generates media based on current Web server traffic thereby optimizing throughput of the media through the Web server.

It would be advantageous to provide a system that generates media that is optimized for the Web client, wherein client connection speed determines optimum quality and file size.

It would be advantageous to provide a system that generates media, whereby the media is automatically uploaded.

It would be advantageous to provide a system that automatically caches generated media so identical requests can be handled without regeneration of images.

It would be advantageous to provide a system that resides behind the Web server, thereby eliminating security issues.

It would be advantageous to provide a system wherein the client browser does not require a plug-in.

It would be advantageous to provide a system wherein the system does not require any changes to a Web server.

It would be advantageous to provide a system wherein the system manages the Web server media cache.

It would be advantageous to provide a system wherein the Web media is generated only if requested by a client browser.

It would be advantageous for a system to reduce the need for a Web author to create different versions of a Web site, the system automatically handling image content.

It would be advantageous to provide dynamic imaging capabilities, have a more complete set of image processing functionality, and be controlled directly through an image URL.

It would be advantageous to provide an end-to-end solution requiring only a standard browser that is completely controllable using the proprietary tags contained within a simple image link in the web page.

It would be advantageous to run an image application as a separate server controlled directly by single image requests to that server, such that any web server, even one that is only sending static HTML can access imaging features.

SUMMARY OF THE INVENTION

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, generated media is cached such that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a schematic diagram showing the placement of the system within a current Web infrastructure according to the invention;

FIG. **2** is a schematic diagram showing how a typical Web site delivers an HTML document and its graphics to a Web browser according to the prior art;

FIG. **3** is a schematic diagram showing delivery of an HTML document and media to a Web browser according to the invention;

FIG. **4** is a schematic diagram showing the components involved in Web site administration according to the prior art;

FIG. **5** is a schematic diagram showing the components of the system involved in Web site administration according to the invention;

FIG. **6** is a simple overview showing the components of the system according to the invention;

FIG. **7** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to the invention;

FIG. **8** is a flow chart showing an authoring process according to the invention;

FIG. **9** is a flow chart showing an HTML parsing process according to the invention;

FIG. **10** is a flow chart showing a media creation process according to the invention;

FIG. **11** is a screen shot showing an administration tool according to the invention;

FIG. **12** displays a structure of a database record used for the system according to the invention;

FIG. **13** shows original media to be processed according to the invention;

FIG. **14** shows a portion on an HTML document with a proprietary tag according to the invention;

FIG. **15** shows an HTML document and an HTML document source according to the invention;

FIG. **16** shows a generated GIF image according to the invention;

FIG. **17** is a schematic diagram of an image system within a typical Web infrastructure according to the invention;

FIG. **18** is a schematic diagram showing delivery of an HTML document and original media according to the invention;

FIG. **19** is a schematic diagram showing components of Web site administration according to a preferred embodiment of the invention;

FIG. **20** is a simple overview showing components of the image system according to a preferred embodiment of the invention;

FIG. **21** is a schematic diagram showing process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention;

FIG. **22** shows a flowchart of a content generation procedure according to a preferred embodiment of the invention; and

FIG. **23** is a flow chart showing an authoring process according to a preferred embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, the generated media is cached so that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

A detailed description of such automatic media delivery system operating in parallel with existing Web site infrastructure is found below in the section under the heading as such.

FIG. **1** is a schematic diagram showing the placement of the system within a current Web infrastructure according to a preferred embodiment of the invention. The system **100** is attached to a Web server **110**, which is connected to multiple client browsers **120**(*a-d*) via the Internet **130**.

FIG. **2** is a schematic diagram showing how a typical Web site delivers an HTML document and its graphics to a Web browser according to the prior art. An original media **200** is passed to post-production systems **210**, wherein the media **200** is manipulated by hand and prepared for the Web. The result is a Web media **220**. The Web media **220** and an associated HTML document **230** referring to the media **220** by media tags are input to a Web server **110** for a Web browser **120** to view via the Internet **130**.

FIG. **3** is a schematic diagram showing delivery of an HTML document and media to a Web browser according to a preferred embodiment of the invention. An original media **200** and an HTML document embedded with proprietary media tags **300** are input into the system **100**. The system **100** generates a Web-safe media **220** and a modified HTML document **230** that refers to the Web media, and automatically loads them onto the Web server **110** for view by a Web browser **120** via the Internet **160**.

FIG. **4** is a schematic diagram showing components involved in Web site administration according to the prior art. Original media assets **400** are original images, video, or sound that have not been prepared for the Web. Web sites usually need to manage the placement of media on the network for easy retrieval by Web designers. Post-production systems **410** vary from Web site to Web site. Post-production systems **410** are usually custom procedures that Web designers use to convert an original media, such as an image, to one that can be displayed on the Web. Post-production systems **410** also upload finished images to Web image systems. Web images **420** are Web versions of the original images. Web images **420** are ready for retrieval by the Web server **110** to be delivered to a Web browser **120**. Any image to be modified or updated must pass through the herein above three components before it can be delivered to the Web browser **120**. HTML pages **460** have references to Web images **420**.

FIG. **5** is a schematic diagram showing the components involved in Web site administration according to a preferred embodiment of the invention. Web site administration is simplified using the claimed invention. Asset management, automatic image manipulation, automatic image conversion, automatic image upload, and automatic disk management **500** are provided by the claimed invention.

FIG. **6** is a simple overview showing the components of the system according to a preferred embodiment of the invention. HTML with proprietary tags **300** is the original HTML document that is embedded with proprietary tags which describe how the images are to be manipulated for the Web. Java servlet engine **600** is a third-party product that allows the system **100** to interface with the Web server **110** and execute Java servlet code. The Web server **110** is third-party software that delivers Web pages to a Browser **120**. The Browser **120** views Web pages that are sent from the Web server **110**. Modified HTML with system created images **230** are a final result of the system. Modified HTML **230** is a standard HTML document without proprietary embedded tags and with standard Web graphics.

The System.

A preferred embodiment of the system **100** is provided.

HTML parsing subsystem **610** parses through an HTML document and searches for proprietary tags. If it finds a proprietary tag it hands it to a media caching subsystem **620** for further processing. The media caching subsystem **620** returns a standard HTML tag. The HTML parsing subsystem **610** then replaces the proprietary tag it found with the returned tag. The parsing subsystem **610** then continues searching for a next proprietary tag, repeating the process herein above. The process is finished when no more proprietary tags can be found.

The media caching subsystem **620** determines if an image has been created for the requested proprietary tag. If the image has already been created and the files that built that image have not been modified, the media caching subsystem **620** returns an HTML tag that refers to a previously-generated image. If the image has not been created, the media caching subsystem **620** hands the HTML tag to a media creation subsystem **630**. The media creation subsystem **630** returns an image to the media caching subsystem **620**. The media caching subsystem **620** adds the created image and the HTML tag to a media cache database **640**.

The media cache database **640** contains references to the created images **645**. In a preferred embodiment, the references are the script used to create the image, the names of the images used to create the image, the dates of those files, and the HTML that represents the created image. The media caching subsystem **620** performs lookups in this database to determine if the image has been created. If the image has not been created the media caching subsystem **620** calls upon the media creation subsystem **630** to create the image and then store the results in the media cache database **640**.

The media creation subsystem **630** takes a proprietary tag from the media caching subsystem **620** and generates an image. The image is generated by deciphering the tag and handing it to the media processing engine **650**. After the image is created, the media creation subsystem returns the name of the newly created image to the media caching subsystem **620**.

The media processing engine **650** interprets the proprietary tag and generates the image. The media processing engine **650** looks up images in a media repository to obtain the location of the original file.

The media repository **660** contains original images **665** used in the system **100**.

FIG. **7** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention. An original media **200** is created. The media **200** is placed into the system **100** in the media repository **660**. Similarly, an HTML document with proprietary tags **300** is created and placed on a Web server **110**. A user requests a Web page from a Web browser **120**. The Web server **110** passes the requested page to an HTML parser **610**. The HTML parser **610** parses HTML looking for media tags. The parser **610** looks up media tags in a media tags database **640**. If the media tag is found, then the system **100** produces a modified HTML document **230**. Otherwise, the media creation subsystem **630** uses the media tag to generate a Web media **220**. The generated Web media **220** is placed in a media cache subsystem **620**. The proprietary media tag is converted by a converter **700** to a standard HTML tag that refers to the generated media **220** in cache. The media tag and the HTML equivalent are stored in the media tags database **640**. Media tags are replaced by standard HTML equivalent to provide a modified HTML document **230**. The modified HTML document **230** is delivered to the Web server **110**. The Web server **100** delivers the modified HTML document **230** to the browser **120** via the Internet for a user to view.

FIG. **8** is a flow chart showing an authoring process according to a preferred embodiment of the invention. The process starts (**800**) when a user adds an original graphic to the system (**810**). The user then creates an HTML document that contains proprietary media tags (**820**). The user then places the HTML document on a Web server (**830**) and ends the authoring process (**840**).

FIG. **9** is a flow chart showing an HTML parsing process according to a preferred embodiment of the invention. The process starts (**900**) when a consumer requests a Web page (**910**). A Web server hands the request of the Web page to the system (**920**). The system parses the Web page (**930**). The system looks for a media tag (**940**). If found, the system retrieves the HTML equivalent of the media tag (**950**) and replaces the media tag with the HTML equivalent tag (**960**). The system continues parsing the Web page for tags (**970**) by returning to step (**940**). When no more tags are found, the system delivers the modified Web page to the Web server (**980**) and therein ends the process (**990**).

FIG. **10** is a flow chart showing a media creation process according to a preferred embodiment of the invention. The process starts (**1000**) when the system requests an HTML equivalent to a proprietary media tag (**1010**). The Media tag is combined with bandwidth information (**1020**). The subsystem checks if the media tag already exists in the media tag database (**1030**). If it does, the subsystem checks if any of the original assets used to create the media have been changed (**1040**). If not, then the subsystem retrieves the HTML equivalent tag from the database (**1050**) and returns the HTML equivalent tag to the requesting system (**1060**). If any of the original assets used to create the media have been changed (**1040**), then the subsystem removes the media tag entry from the media database (**1070**) and creates the media using the media tag (**1080**). The subsystem then stores the media in a media cache (**1090**). The subsystem generates the HTML referring to the generated media (**1100**) and places the media tag and the HTML equivalent in the media tag database (**1110**). The HTML equivalent is returned to the requesting system (**1060**) and the process stops (**1120**).

The differences between using HTML and the proprietary tags disclosed herein are noted. HTML allows Web designers to create Web page layouts. HTML offers some control of the images. HTML allows the Web designer to set the height and width of an image. However, all of the other image operations disclosed herein are supported by the claimed invention and are not supported by HTML.

Table A herein below provides the claimed proprietary tags according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

TABLE A

Tags
Generate image
<freerideimage> mediascript </freerideimage>
Generate a standard Web image.
Generate thumbnail image linked to full image
<freerideimagethumbnail> mediascript <xs=size ys=size /freerideimagethumbnail>
Generate a thumbnail of specified size and link it to the full size version.
Generate zoom and pan image
<freerideimagezoom> mediascript </freerideimagezoom>
Generate a zoomable/panable image.
Security
<freerideimagesecure> </freerideimagesecure>
Specifies that all images found between these tags are secured images and the system will determine access before generating.

Table B herein below provides the claimed script commands according to a preferred embodiment of the invention. Additional commands may be added as needed.

TABLE B

Media processing script commands
Add Noise
Noise_AddNoise([amount=<value 1..999>] [gaussian] [grayscale])
This command adds noise to the image.
Adjust HSB
AdjustHsb([hue @ <value ±255>] [saturation @ <value ±255>] [brightness @ <value ±255>])
This command allows the HSB of an image to be altered. This can be applied to images of all supported bit-depths.
Adjust RGB
AdjustRgb([brightness @ <value ±255>] [contrast @ <value ±255>] [red @ <value ±255>] [green @ <value ±255>] [blue @ <value ±255>] [noclip @ <true, false>] [invert @ <true, false>])
This command allows the contrast, brightness, and color balance of an image to be altered.
Blur
Blur(radius @ <value 0..30>)
This command applies a simple blur filter on the image.

TABLE B-continued

Media processing script commands
Blur Convolve
Blur__Blur( )
This command commands perform a simple 3 × 3 convolution for blurring.
Blur Convolve More
Blur__MoreBlur( )
This command commands perform a stronger 3 × 3 convolution for blurring.
Blur Gaussian
Blur__GaussianBlur([radius=<value 0.1..250>])
This command applies a Gaussian blur to the image.
Blur Motion
Blur__MotionBlur([distance=<value 1..250>] [angle=<degrees>])
This command applies motion blurring to the image using the specified distance and angle.
Brush Composite
Composite(source @ {<User-Defined Media Object name>} [x @ <pixel>] [y @ <pixel>]
[onto] [opacity @ <value 0..255>] [color @ <color in hexadecimal>] [colorize @ <true, false>]
[saturation @ <value 0..255>])
This command composites the specified “brush” (foreground) image onto the current “target” (background) image.
Colorize
Colorize(color @ <color in hexadecimal> [saturation @ <value 0..255>])
This command changes the hue of the pixels in the image to the specified color.
Convert
Convert(rtype @ <bit-depth> {dither @ <value 0..10>])
This command converts the image to the specified type/bit-depth.
Convolve
Convolve(Filter @ <filtername>)
This command applies a basic convolution filter to the image. In a user interface driven system, the filters could be stored in files and edited/created by the user.
Crop/Resize Canvas
Crop([xs @ {<pixels>, percentage + “%”>}] [ys @ {<pixels>, percentage + “%”>}] [xo @ <left pixel>]
[yo @ <top pixel>] [padcolor @ <color in hexadecimal>] [padindex @ <value 0..255>])
This command crops the media to a specified size.
Discard
Discard( )
This command removes the designated Media Object from memory.
Drop Shadow
DropShadow([dx @ <pixels>] [dy @ <pixels>] [color @ <color in hexadecimal>] [opacity @ <value 0..255>] [blur @ <value 0..30>] [enlarge @ <true, false>])
This command adds a drop shadow to the image based on its alpha channel.
Equal
Equal(source @ {<User-Defined Media Object name>})
This command compares the current media with the one specified. If the media are different in any way, an error value is returned.
Equalize
Equalize([brightness @ <−1, 0..20>] [saturation @ <−1, 0..20>])
This command equalizes the relevant components of the media. Equalization takes the used range of a component and expands it to fill the available range.
Export Channel
ExportGun(Channel @ <channelname>)
This command exports a single channel of the source as a grayscale image.
Find Edges
Stylize__FindEdges([threshold=<value 0..255>] [grayscale] [mono] [invert])
This command finds the edges of the image based on the specified threshold value.
Fix Alpha
FixAlpha( )
This command adjusts the RGB components of an image relative to its alpha channel.
Flip
Flip(<horizontal, vertical> @ <true, false>)
This command flips the media vertically or horizontally.
Frame Add
FrameAdd(Source @ <filename>)
This command adds the given frame(s) to the specified Media Object.
Glow/Halo
Glow(Size @ <value 0..30> [halo @ <value 0..size>] [color @ <color in hexadecimal>]
[opacity @ <value 0..255>] [blur @ <value 0..30>] [enlarge @ <true, false>])
This command produces a glow or halo around the image based on the image’s alpha.
High Pass
Other__HighPass([radius=<value 0.1..250>])
This command replaces each pixel with the difference between the original pixel and a Gaussian blurred version of the image.
Import Channel
ImportGun(channel @ <channel name> source @ {<User-Defined Media Object name>}
[rtype @ <bit-depth>])
This command imports the specified source image (treated as a grayscale) and replaces the selected channel in the original.
Load

TABLE B-continued

Media processing script commands
Load(Name @ <filename> [type @ <typename>] [transform @ <true, false>])
This command loads a media from the specified file.
Maximum
Other__Maximum([radius=<value 1..10>])
This command scans the area specified by the radius surrounding each pixel, and then replaces the pixel with the brightest pixel found.
Minimum
Other__Minimum([radius=<value 1..10>])
This command scans the area specified by the radius surrounding each pixel, and then replaces the pixel with the darkest pixel found.
Normalize
Normalize([clip @ <value 0..20>])
This command expands the volume of the sample to the maximum possible.
Pixellate Mosaic
Pixellate__Mosaic([size=<value 2..64>])
This command converts the image to squares of the specified size, where each square contains the average color for that part of the image.
Pixellate Fragment
Pixellate__Fragment([radius=<value 1..16>])
This command produces four copies of the image displaced in each direction (up, down, left, right) by the specified radius distance and then averages them together.
Quad Warp
QuadWarp([tlx=<position>] [tly=<position>] [trx=<position>] [try=<position>] [blx=<position>] [bly=<position>] [brx=<position>] [bry=<position>] [smooth])
This command takes the corners of the source image and moves them to the specified locations, producing a warped effect on the image.
Reduce to Palette
Reduce([colors @ <num colors>] [netscape @ <true, false>] [b&w @ <true, false>] [dither @ <value 0..10>] [dithertop @ <value 0..10>] [notbackcolor] [pad @ <true, false>])
This command applies a specified or generated palette to the image.
Rotate
Rotate(Angle @ <value 0..359> [smooth @ <true, false>] [enlarge @ <true, false>] [xs @ <pixels>]
[ys @ <pixels>])
This command rotates the media by the specified angle in degrees.
Rotate 3D
Rotate3d([anglex @ <angle ±89>] [angley @ <angle ±89>] [distance @ <value>])
This command rotates the image in 3D about either the x-axis or y-axis.
Save
Save([type @ <image-type>])
This command saves a media to the specified file.
Scale
Scale([xs @ {<pixels>, <percentage + "%">}] [ys @ {<pixels>, <percentage + "%">}] [constrain @ <true, false>] [alg @ {"fast", "smooth", "outline"}] [x1 @ <pixels>] [y1 @ <pixels>] [x2 @ <pixels>] [y2 @ <pixels>])
This command scales the image to the specified size.
Select
Selection([source @ <User-Defined media Object>}] [remove @ <true, false>] [invert @ <true, false>]
[backcolor] [color=<color>] [index=<value>] [opacity @ <value 0..255>])
This command manages the selected region for the current Media Object.
Set Color
SetColor([backcolor @ <color in hexadecimal>] [forecolor @ <color in hexadecimal>] [backindex @ <value 0..255>] [foreindex @ <value 0..255>] [transparency @ ("on","off")])
This command allows the background color, foreground color, and transparency state of an image to be set.
Set Resolution
SetResolution([dpi @ <value>] [xdpi @ <value>] [ydpi @ <value>])
This command changes the DPI of the image in memory.
Sharpen
Sharpen__Sharpen( )
This command sharpens the image by enhancing the high-frequency component of the image.
Sharpen More
Sharpen__SharpenMore( )
This command sharpens the image by enhancing the high-frequency component of the image, but is stronger than the standard sharpening.
Stylize Diffuse
Stylize__Diffuse([radius=<value 0..>] [lighten] [darken])
This command diffuses the image by randomizing the pixels within a given pixel radius.
Stylize Embose
Stylize__Emboss([height=<value 1..10>] [angle=<degrees>] [amount=<percentage 1..500>])
This command converts the image to an embossed version.
Text Drawing
DrawText(Text @ <string> Font @ <font file> [size @ <value>]
[color @ <color in hexadecimal>] [smooth @ <true, false>] [<left, right, top, bottom> @ <true, false>]
[x @ <pixel>] [y @ <pixel>] [wrap @ <pixel-width>] [justify @ {left,center,right}] [angle @ <angle>])

TABLE B-continued

Media processing script commands
This command composites the specified text string onto the image.
Text Making
MakeText(text @ <string> font @ <font file> [path @ <path to font directory>] [size @ <value 1..4095>]
[color @ <color in hexadecimal>] [smooth @ <true, false>] [wrap @ <pixel-width>]
[justify @ {left,center,right}] [angle @ <angle>])
This command creates a new image that includes only the specified text.
Trace Contour
Stylize__TraceContour([level=<value 0..255>] [upper] [invert])
This command traces the contour of the image at the specified level (for each gun).
Unsharpen Mask
Sharpen__UnsharpMask([amount=<percentage 1..500] [radius=<value 0.1..250>] [threshold=<value 0..255>])
This command enhances the edges and detail of an image by exaggerating differences between the image and a gaussian blurred version of the same image.
Zoom
Zoom([xs @ <pixels>] [ys @ <pixels>] [scale @ <value>] [x @ <left pixel>] [y @ <top pixel>])
This command zooms in on a specified portion of the media and fits it to the specified size. This constitutes a crop followed by a scale.

Table C herein below provides a list of features provided by a preferred embodiment of the invention. It is noted that the list of features included in Table C is by no means complete. In other embodiments, the list of features is expanded or reduced as needed.

TABLE C

System Feature List
Reads and writes various file formats:
BMP, GIF, JPG, PNG, TIF, PICT, TGA, PSD, FPX;
Supports many image processing operations;
Dynamically creates Web images from original assets;
Dynamically creates thumbnail images;
Dynamically creates images that can be panned and zoomed without browser plug-ins or special file formats;
Automatically propagates changes of original assets throughout a Web site;
Uses an intelligent caching mechanism:
Clean up image cache on demand;
Eliminates orphaned image files; and
Optimizes Web server cache by providing most recent images;
Renders TrueType fonts on the server instead of browser;
Uses intelligent scaling of line drawings;
Allows Web designers to manipulate images with proprietary tags;
Preserves original image assets;
Optimizes Web server traffic by adjusting the bandwidth of graphics;
Optimizes images for client connection speed;
Allows clients to specify the quality of images on a Web site; and
Allows Web designers to dynamically create images by manipulating proprietary tags in their applications (server or client side).

FIG. **11** is a screen shot showing an administration tool according to a preferred embodiment of the invention. Specifically, FIG. **11** shows an administration page that contains cached images of generated scripts. The use of the term “fre-eride” refers to an internal code name for the invention.

FIG. **12** displays a structure of a database record used for the system according to a preferred embodiment of the invention. A Script Table **1200** has 5 columns, Media Script **1210**, HTML Equivalent **1220**, Bandwidth **1230**, Generated File **1240**, and Dependency List **1250**. A Dependency Table **1260** has two columns, File Name **1270** and Modification Date **1280**.

Snowboard Store Example.

Background.

The snowboard store highlights several features of the claimed system. The snowboard store is an imaginary store that allows a user to configure his or her snowboard. The store consists of five logos, five board colors, and four boards. The consumer clicks on the buttons to change the snowboard represented in the middle of the screen. When the consumer has configured the snowboard they the snowboard can be purchased by selecting a buy button.

Prior Art Method.

To create the snowboard site today, the Web designer must render all possible combinations of the board. The number of combinations is five logos×five board colors×four boards=100. The designer also must render all the buttons. The creation process is very tedious and involves a lot of production work. Typically, most Web sites do not even attempt such an endeavor. Also, other issues must be addressed, such as, for example, updating the Web site and scripting. For example, updating a single logo involves updating a minimum of 20 images.

The prior art method sustains a graphic intensive site that requires management of at least 100 images. Updates to the Web site are time-consuming and prone to human error.

The Claimed Method.

A preferred embodiment of the method scripts the image creation process in HTML to create a dynamic Web site. There is no need to create over 100 images. The claimed system generates images on demand. The Web site only needs to create original assets. The scripting process involves writing the proprietary scripts. In the current example herein, scripting buttons is very simple. Once one button is created, simply copy and paste the HTML to create another button or many buttons. Only the name of the image to be overlaid on the button must be changed. The Webmaster then creates a simple program that reads what object a user has clicked on and generates a proprietary tag. The tag is then sent to the claimed system to generate a center image.

The claimed method allows the creation of all 100 combinations automatically. When the Web site receives an updated image, only the original image needs to be updated. Any change to the original image automatically propagates throughout the system. The Web site is easier to manage. Testing of the Web site is easier because there is no need to test all 100 combinations. A small subset of combinations will guarantee adequate coverage.

Processing of an Image Tag Example (FIG. **13-16**).

FIG. **13** shows two original images **1300** and **1310** to be processed according to a preferred embodiment of the invention.

FIG. **14** shows a portion on an HTML document with a proprietary tag **1400**, <freerideimage></freerideimage> according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **15** shows an HTML document **1500** as viewed in a browser and an HTML document source **1510**, according to a preferred embodiment of the invention. The use of the term "freeride" refers to an internal code name for the invention.

FIG. **16** shows a generated GIF image **1600** according to a preferred embodiment of the invention.

Automatic Media Delivery System Operating in Parallel with Existing Web Site Infrastructure.

It should be noted that the words, media, graphics, and images are used herein interchangeably.

An automatic graphics delivery system that operates in parallel with an existing Web site infrastructure is provided. The system streamlines the post-production process by automating the production of media through content generation procedures controlled by proprietary tags placed within URLs embedded within Web documents. The author simply places the original media in the system, and adds proprietary tags to the URLs for accessing that media. The system automatically processes the URL encoded tags and automatically produces derivative media for the web site from the original media.

The system takes as input the client connection, server traffic, content generation procedures, and proprietary tags placed within the URL to generate optimized media for the client. The need for the Web author to create different versions of a Web site is reduced because the image content of the site is automatically handled by the system. In addition, the generated media is cached so that further requests for the same media require little overhead.

Because the invention takes the original media, content generation procedures, and proprietary URL tags as inputs for generating the Web media, it is possible to modify any of these inputs and have the system automatically update the media on the associated Web pages.

A preferred embodiment of the invention is described with reference to FIG. **17**. FIG. **17** is a schematic diagram of an image system within a typical Web infrastructure according to the invention. The image system **100** is placed in parallel to an existing Web server **110**. The image system **100** may be on-site or off-site to the Web infrastructure. Multiple client browsers **120***a*-**120***d* communicate with both the Web server **110** and the image server **100** via the Internet **130**.

The delivery of an HTML document and media according to a preferred embodiment of the invention is described with reference to FIG. **18**. Resource locators (URLs) are placed within HTML documents **301** accessible to the Web server **110**. These URLs direct browsers to generate requests for media to the system **100**. The system processes such URLs by interpreting the proprietary tags, executing the indicated image generation procedures on the original media **200**, and returning derivative Web-safe media to client browsers **120***a*-**120***d* via the Internet **130**. Additionally, such generated media is cached on the image server **100** and, therefore need not be regenerated for subsequent requests.

Web site administration according to another preferred embodiment of the invention is described with reference to FIG. **19**. FIG. **19** is a schematic diagram showing the components of Web site administration according to a preferred embodiment of the invention, whereby Web site administration is simplified. The preferred embodiment provides, but is not limited to the following services: asset management, automatic image manipulation, automatic image conversion, automatic image upload, automatic image customization based on browser characteristics, automatic disk management, automatic control of proxy caching, and image delivery **501**.

FIG. **20** is a simple overview showing components of the system according to a preferred embodiment of the invention. HTML pages with proprietary URL tags **301** describe how referenced media therein is to be manipulated for Web. Browsers **120** send such tags to the image system **100** as media requests. A server **2000** within the image system **100** receives the media requests, decodes the URL tags, and retrieves any media that already exists in the media caching system **2010**. Non-existent media is subsequently generated by a media creation system **2020** using original media **2050** stored in a media repository **2040** and using content generation procedures **2030**.

The image System.

Following is a detailed description of the preferred embodiment of the invention with reference to FIG. **21** below.

The system receives a request for media through a URL containing proprietary tags for controlling image generation. The system parses this URL to determine the content generation procedure to execute, input to the content generation procedure, post-processing directives for, for example, zoom/pan/slice, browser properties, and any cache control directives. Such data is handed to a media caching subsystem that returns the requested image if found. If the image is not found, the information is handed to the media generation subsystem that executes the specified content generation procedure to produce a derivative image. The media generation subsystem returns one or more images to the media caching system for subsequent reuse.

The media caching subsystem is a mechanism for associating final or intermediate derivative media with the procedure, input, and user characteristics used to generate said media, specified through proprietary tags within the requested URL. This system may be implemented using a database, file system, or any other mechanism having capability to track such associations.

The media generation subsystem executes a primary content generation procedure to produce a derivative image whose identifier is provided to the media caching subsystem. This derivative image is composed of one or more original images acquired from the media repository. This media is then passed to the dynamic image content system, if necessary, to generate a subsequent derivative media suitably modified for the needs of zooming, panning, or slice. The resulting media is passed to the user profile system where it is again modified to account for any specific user browser characteristics specified using the proprietary URL tags. This media is then returned to the browser, along with any cache control directives encoded within the URL, and its identifier is passed to the media caching system for subsequent retrieval.

The dynamic content system operates on intermediate derivative images to generate image subsets or scalings used by Web site designers to implement zooming in on an image, panning across an image, slicing an image into parts, and the like for special Web page effects. The input to this system is cached by the media caching system such that the intermediate image need not be regenerated.

The user profile system operates on the final image about to be returned to the browser and may modify the image to account for individual needs of Web site users. The designer of a site is able to implement freely custom post-processing of images to meet the specific needs of their clients.

FIG. **21** is a schematic diagram showing the process flow of a proprietary enabled page delivered to a Web browser according to a preferred embodiment of the invention. Original media **200** is created and placed into the system **100** in a media repository **2040**. A content generation procedure **2140** is created with instructions on how the media is to be transformed to create the desired Web page content. An HTML page **301** is created for the Web site comprising the system **100**, the page containing one or more URLs directing a browser **120** to request the specified content generation procedure **2140** from the system **100** using input parameters specified with proprietary tags encoded within the URL. The browser **120** requests the Web page **301** from the Web site **110**. Upon receipt of the page **301**, the browser contacts the system **100** requesting media specified in the URL. The system parses the URL **2100** to determine the content generation procedure **2140** to execute, any corresponding input parameters to be used by such procedure, any dynamic content processing **2150** to be performed by dynamic media procedures, any user profile information **2160** to be used to modify the resulting image, and any cache control HTTP headers **2190** the site instructs to accompany the resulting image.

The parser generates a unique primary lookup key **2110** for the specified resulting media. If the key corresponds to an existing generated media **2180**, such media is returned immediately to the browser **120** through a media cache **2120**, and the transaction is complete. Otherwise, a media generation occurs. In the case of media generation requiring dynamic content processing, a unique secondary lookup key corresponding to intermediate media is generated **2130**. If intermediate media **2170** corresponding to this key is found, such media is passed directly to the dynamic media content system **2150** having dynamic media procedures, wherein appropriate action is taken to generate the required derivative from the intermediate media data. A unique key is generated for the derivative **2130** and passed to the media caching system **2120**. If the media caching system finds no such intermediate image, such intermediate image is generated according to instructions specified by the content generation procedure, cached by the media cache system **2120** as a secondary cached media **2170**, and passed to the dynamic media system **2150**. Again, appropriate action is taken to generate the required derivative from the intermediate image data.

The resulting image after any dynamic media processing is complete, is checked to ensure that the image is in a valid Web image format. If not, the image is automatically converted into a valid format.

The final media is passed to a user profile system **2160** wherein browser characteristics specified through proprietary tags within the URL are inspected, and appropriate modification to the media is performed, based on such characteristics. The resulting image is handed to the media cache system **2120** for caching and returned to the browser **120**.

FIG. **22** shows a flowchart of the content generation procedure according to a preferred embodiment of the invention. A URL containing proprietary tags (**2200**) is parsed (**2210**) to determine the content generation procedure to execute, any dynamic modifications to the media, user profile characteristics, and proxy-cache control. A unique final lookup key is generated for the media (**2220**) and the media cache is checked (**2230**). If the indicated media exists, control passes to proxy-cache control (**2290**) and the media is delivered to the browser (**2295**). Otherwise, dynamic media system tags are separated from content generation control tags (**2240**) and a unique intermediate image lookup key is generated (**2250**). The cache is then checked for such intermediate media (**2261**). If such intermediate media is found, it is used directly for dynamic processing, if required. Otherwise, content is generated (**2262**) and cached (**2263**), and the result is evaluated for dynamic processing (**2270**). If dynamic processing is required, the media is operated upon by the dynamic content generator (**2271**), otherwise it is evaluated for valid content type (**2272**). If the content type is invalid, the media is automatically converted to a valid type (**2273**). The resulting image is then customized by the user profiling system (**2280**) for specified browser or client attributes. Finally, any cache-control directives specified are attached to the response (**2290**) and the media is delivered to the browser (**2295**).

FIG. **23** is a flow chart showing an authoring process according to a preferred embodiment of the invention. The process starts (**2300**) when a user adds an original graphic or other media (**2310**) to the system. The author then creates a content generation procedure (**2320**) containing instruction on how the original media should be processed to generate the desired Web page content. The user then creates an HTML document (**2330**) that refers to that image by using a URL pointing to a content generation procedure on the image server. The system ends (**2340**). The authoring subsystem assists the Web site designer with choosing parameters and with designing the content generation procedure such that the desired Web site graphic is obtained.

It should be appreciated that differences exist between specifying an image with a URL and requesting an image using a content creation process that interprets proprietary parameters encoded within a URL. That is, URLs allow Web site designers to load specific graphic images into a Web page. In contrast and according to the invention, URLs containing proprietary content creation tags initiate a process whereby graphic images for a site are automatically produced.

Table D below is a list of example proprietary URL tags used for content generation within the system according to the preferred embodiment of the invention. Additional tags may be added to the system as necessary.

TABLE D

Tags
f=function
Names the content creation procedure used to generate all or part of the desired graphic.
args=arguments
Supplies page dependent parameters used to control the content creation procedure from within the Web page.
cr=crop rectangle
Specifies that portion of the image generated by the content generation procedure to be returned to the browser.
st=slice table
Specifies a rectangular grid to be placed over the image produced by the content generation procedure, each portion of which can be returned to the browser.
sp= slice position

TABLE D-continued

Tags
Specifies that portion of the slice table grid placed over the image generated by the content creation procedure to be returned to the browser.
is=image size parameter
Specifies scale factors to be applied to any portion of an image generated by any combination of a content generation procedure, arguments, crop rectangles, slice tables, and slice positions.
p=user profile string
Specifies a user profile identifier used to modify the final image prior to returning the image to the browser, thus allowing clients to modify the image returned to the browser to account for individual browsing conditions.
c=cache control
Specifies a proxy-cache control string to accompany the returned image within an HTTP header.

Table E below is a list of example supported content creation commands according to a preferred embodiment of the invention. Additional commands may be added as necessary.

TABLE E

Content Creation Commands
Adjust HSB
Allows the HSB of an image to be altered.
Adjust RGB
Allows the contrast, brightness, and color balance of an image to be altered.
Colorize
Alters the hue of the pixels in the image to that of the specified color.
Brush Composite
Composites the specified brush image onto the current target image.
Convert
Converts the rasters to the specified type/bit-depth.
Crop
Crops the media to the specified size.
Dropshadow
Adds a drop shadow to the image, based on the alpha-channel.
Equalize
Performs an equalization on the relevant components of the media.
FixAlpha
Adjusts the RGB components of an image relative to its alpha-channel.
Flip
Flips the media vertically or horizontally.
Glow
Produces a glow or halo around the image.

TABLE E-continued

Content Creation Commands
Load
Loads in a media from the specified file.
Normalize
Similar to equalize, but for audio.
Reduce
Reduces the image to a specified palette.
Rotate
Rotates the media clockwise by the specified angle in degrees.
Save
Saves the media to the specified file.
Scale
Scales the media to the specified size.
SetColor
Allows the background color, foreground color, and transparency state of the media to be set.
Text Drawing
Composites the specified text onto the image.
Text Making
This command, instead of compositing text onto the target, creates a new image that just encloses the text.
Zoom
Zooms in on a specified portion of the media, and fits it to the specified size. Effectively this constitutes a crop followed by a scale.

Table F below lists comprises some, and is not limited to all major features of a preferred embodiment of the invention. Additional features may be added as necessary.

TABLE F

System Features
Reads and writes various file formats;
Supports many image processing operations;
Dynamically creates Web images from original assets;
Dynamically creates thumbnail images;
Dynamically and efficiently creates images that can be panned, zoomed, or sliced from original assets without Browser plugins;
Automatically propagates changes in original assets throughout the Web site;
Uses an intelligent caching mechanism for both final and intermediate graphics, comprising:
Clean up cache on demand;
Eliminates orphaned Web files; and
Optimizes Web server cache by providing most recent images;
Renders True-Type fonts on server instead of browser;
Uses intelligent scaling of line drawings;
Allows Web designers to manipulate images using a combination of content generation procedures and proprietary URL tags;
Preserves original image assets;
Optimizes Web server traffic by adjusting the bandwidth of graphics;
Optimizes images for client connection speed;
Allows clients to specify the quality of images on a Web site;
Allow site-specific customized image optimizations for a variety of purposes; and
Allows Web designers to dynamically create images by manipulating proprietary URL tags in applications.

Accordingly, although the invention has been described in detail with reference to a particular preferred embodiment, persons possessing ordinary skill in the art to which this invention pertains will appreciate that various modifications and enhancements may be made without departing from the spirit and scope of the claims that follow.

The invention claimed is:

1. A method in a host computer for developing transformation processing operations to optimize media content playback to a plurality of playback devices connected with the host computer in a network, the method comprising:

receiving a first request from a first playback device for media content;

wherein the first request contains information, the information indicating a first original media content, first content generation operations, and first transformation operations;

determining whether a previously-generated first intermediate media content is available for reuse, the previously-generated first intermediate media content having been created using the first original media content and the first set of content generation operations; and

responsive to determining that a previously-generated first intermediate media content is available, creating a first optimized media content for the first playback device by performing the first set of transformation operations on the previously-generated first intermediate media content; and

responsive to determining that a previously-generated first intermediate media content is not available, creating a first optimized media content for the first playback device by creating a first intermediate content using the first original media content and the first set of content generation operations, and performing the first set of transformation operations on the first intermediate media content; and

sending the first optimized media content to the first playback device.

2. The method of claim **1**, further comprising:

receiving a second request from a second playback device for media content;

wherein the second request contains information, the information indicating a second original media content, second set of content generation operations, and second set of transformation operations;

wherein the second request is received substantially concurrently with receiving the first request;

wherein the first original media content and the second original media content are the same media content;

determining whether a second intermediate media content is available having been previously created using the second original media content and the second set of content generation operations; and

responsive to determining that the second intermediate media content is available, creating a second optimized media content for the first playback device by performing the second set of transformation operations on the second intermediate media content; and

responsive to determining that the second intermediate media content is not available, creating a second optimized media content for the first playback device by creating the second intermediate content using the second original media content and the second set of content generation operations, and performing the second set of transformation operations on the second intermediate media content; and

sending the second optimized media content to the second playback device.

3. The method of claim **2**, wherein the first set of content generation operations and the second set of content generation operations are the same, and the first set of transformation operations and the second set of transformation operations are different.

4. The method of claim **2**, wherein the first set of content generation operations and the second set of content generation operations are different, and the first set of transformation operations and the second set of transformation operations are different.

5. The method of claim **1**, further comprising:

determining whether a previously-generated first optimized media content is available for reuse, wherein the previously-generated first optimized media content was created using the first original media content, the first content generation operations, and the first transformation operations;

responsive to determining that the previously-generated first optimized media content is available, sending the previously-generated first optimized media content to the first playback device.

6. The method of claim **1**, further comprising:

determining whether the host computer has sufficient processing resources to create the first optimized media content for the first playback device;

responsive to determining that the host computer does not have sufficient processing resources to create the first optimized media content, determining an alternate first set of content generation operations or an alternate first set of transformation operations;

creating an alternate first optimized media content using the alternate first set of content generation operations or the alternate first set of transformation operations.

7. The method of claim **6**, wherein the first optimized media content is at a first level of quality and the alternate first optimized media content is at a second level of quality, wherein the first level of quality is higher than the second level of quality;

wherein a level of quality of a media content is measured based on a compression format, a bit rate, and an image resolution of a media content sent to a playback device.

8. A non-transitory computer-readable memory storing instructions, the instructions which when executed by a processor, cause the processor to perform:

receiving a first request from a first playback device for media content;

wherein the first request contains information, the information indicating a first original media content, first content generation operations, and first transformation operations;

determining whether a previously-generated first intermediate media content is available for reuse, the previously-generated first intermediate media content having been created using the first original media content and the first set of content generation operations; and

responsive to determining that a previously-generated first intermediate media content is available, creating a first optimized media content for the first playback device by performing the first set of transformation operations on the previously-generated first intermediate media content; and

responsive to determining that a previously-generated first intermediate media content is not available, creating a first optimized media content for the first playback device by creating a first intermediate content using the first original media content and the first set of content generation operations, and performing the first set of transformation operations on the first intermediate media content; and

sending the first optimized media content to the first playback device.

9. The non-transitory computer-readable memory of claim **8**, further comprising:

receiving a second request from a second playback device for media content;

wherein the second request contains information, the information indicating a second original media content, second set of content generation operations, and second set of transformation operations;

wherein the second request is received substantially concurrently with receiving the first request;

wherein the first original media content and the second original media content are the same media content;

determining whether a second intermediate media content is available having been previously created using the second original media content and the second set of content generation operations; and

responsive to determining that the second intermediate media content is available, creating a second optimized media content for the first playback device by performing the second set of transformation operations on the second intermediate media content; and

responsive to determining that the second intermediate media content is not available, creating a second optimized media content for the first playback device by creating the second intermediate content using the second original media content and the second set of content generation operations, and performing the second set of transformation operations on the second intermediate media content; and

sending the second optimized media content to the second playback device.

10. The non-transitory computer-readable memory of claim **9**, wherein the first set of content generation operations and the second set of content generation operations are the same, and the first set of transformation operations and the second set of transformation operations are different.

11. The non-transitory computer-readable memory of claim **9**, wherein the first set of content generation operations and the second set of content generation operations are different, and the first set of transformation operations and the second set of transformation operations are different.

12. The non-transitory computer-readable memory of claim **8**, further comprising:

determining whether a previously-generated first optimized media content is available for reuse, wherein the previously-generated first optimized media content was created using the first original media content, the first content generation operations, and the first transformation operations;

responsive to determining that the previously-generated first optimized media content is available, sending the previously-generated first optimized media content to the first playback device.

13. The non-transitory computer-readable memory of claim **8**, further comprising:

determining whether the host computer has sufficient processing resources to create the first optimized media content for the first playback device;

responsive to determining that the host computer does not have sufficient processing resources to create the first optimized media content, determining an alternate first set of content generation operations or an alternate first set of transformation operations;

creating an alternate first optimized media content using the alternate first set of content generation operations or the alternate first set of transformation operations.

14. The non-transitory computer-readable memory of claim **13**, wherein the first optimized media content is at a first level of quality and the alternate first optimized media content is at a second level of quality, wherein the first level of quality is higher than the second level of quality;

wherein a level of quality of a media content is measured based on a compression format, a bit rate, and an image resolution of a media content sent to a playback device.

* * * * *

Exhibit G

Source: https://eqwebmgen.eqnetwork.com/merchandizer/
unizoom-multipage/?docURL=/mrex/uploads/
EQwebgraphics/brochures/generated/220dpi/
MediaRichBrochureV6.tif&page=1&docURL=/mrex/
UploadAndPublish/uploads/EQwebgraphics/brochures/
MediaRichBrochureV6.pdf

MEDIARICH® SERVER equilibrium

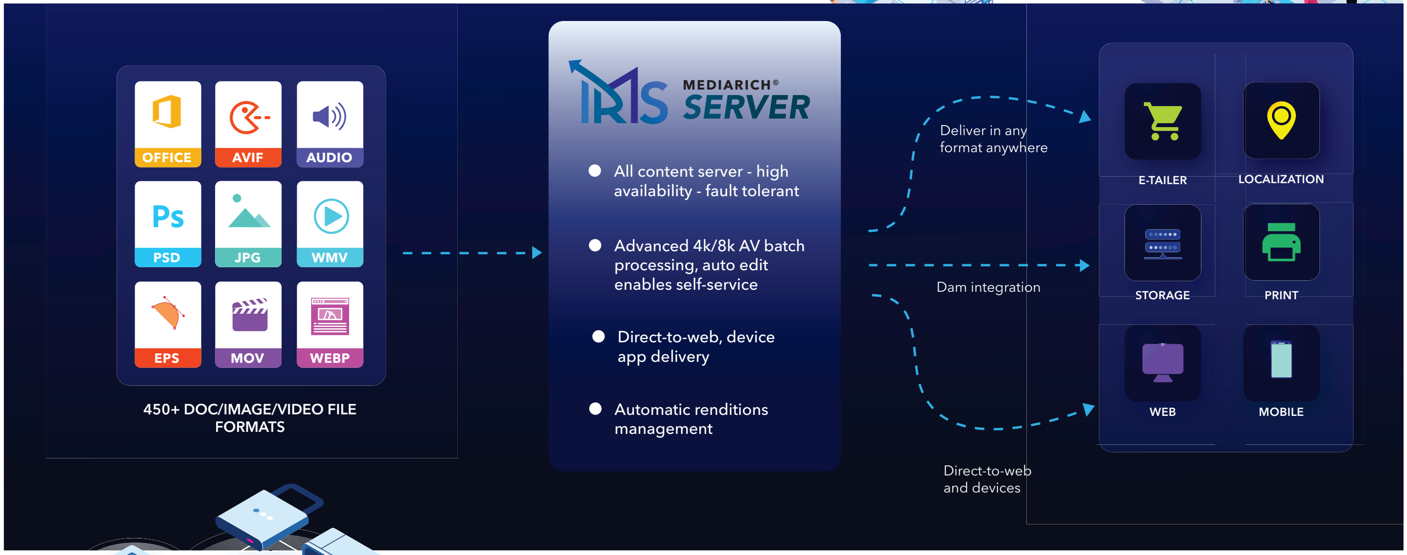
MediaRich 6 is a breakthrough 64bit native content processor

When you need automatic editing of content at scale, MediaRich Server is the de-facto standard for on-the-fly asset generation from any original content source.

MediaRich is the original on-premise or virtual hosted dynamic content processor for any high availability or content self-service applications. MediaRich Server streamlines production workflows by enabling delivery to virtually any Web site, app, or device from inside your digital, web, video, media, or enterprise asset management system.

MediaRich complements existing technologies and interfaces with any existing infrastructure accelerating time-to-market for publishing rich visual content. Now with OneViewer™ Mobile/AEC, MediaRich enables next-generation, view everywhere dynamic mobile experiences for all content. This provides high-quality viewing with low bandwidth for any kind of media, automatically.

MediaRich 64bit is now Azure , Amazon EC2 or other data center ready.



450+ DOC/IMAGE/VIDEO FILE FORMATS

MEDIARICH® SERVER

- All content server - high availability - fault tolerant
- Advanced 4k/8k AV batch processing, auto edit enables self-service
- Direct-to-web, device app delivery
- Automatic renditions management

Deliver in any format anywhere

Dam integration

Direct-to-web and devices

Key Features

For Creatives, Marketing, and IT Professionals

- 450+ camera, image, Office, drawing, and video formats
- Multi-Page zoom & pan support enables view everywhere
- Full Audio/Video support including 4k/8k HD & Broadcast
- High-speed parallel video transcoding and auto-assembly
- CMYK with ICC profile color management for any OS
- Fast loading without sacrificing image quality
- Dynamically generate beautiful text from TT & PS fonts
- Dynamically generate charts and graphs from databases
- Dynamically change colors of images/ layers in RGB, CMYK, hex
- Customize with MediaScripts to fit your IT infrastructure & production environments
- Integrates with Sharepoint 2013/2016/2019, North Plains, NetXposure, Censhare, HP Portal Services, Drupal, Be the Brand Experience and other DAM's
- Digimarc® pre-integrated forensic watermarking
- Full metadata support for IPTC, Exif & XMP Standards
- APIs for web services, .NET C#, COM & cross-platform Java
- Available for Windows, OS X, and Linux
- Certified with Azure and Amazon virtual instances

IRIS MEDIARICH® SERVER

Solutions

Creative, development, and IT professionals working on the biggest B2B portals, eTail sites, media and entertainment companies, and corporate communications rely on MediaRich to automate pdf, image, video processing, streamlining their workflows, and enabling instant mobility experiences.

Corporate control - meets self-service

MediaRich is a critical component of successful Enterprise Content Management. MediaRich integrates easily with custom and commercial digital and media asset management systems creating a brand asset management solution

Make your brand work harder, not your art department

Leverage your brand assets more effectively when your team can access and perform repetitive tasks without hammering your art department. Modify & distribute advertising, tradeshow graphics, and collateral while maintaining brand consistency

Derivatives that work - one image to any size, format or resolution

From a single original image or video file, MediaRich Processes derivatives - in any size, format, or resolution - in a significantly shorter time and at a lower cost compared to a manual workflows.

Cut production time and file system maintenance

Save time by pre-processing derivative images and videos.Decrease file system maintenance by the Web Department. MediaRich increases the quantity and, speed at which images and videos can be made available to internal customers, affiliate partners, and advertising agencies online.

Repurposing on demand

MediaRich automates image and video processing, enabling syndication of rich media assets across an entire organization and beyond to any device, anywhere. MediaRich enables businesses to dynamically repurpose marketing collateral and promotional materials for use in online catalogs, print, or multimedia, and mobile devices, as well as for use on affiliate Web sites, intranets, extranets, and portals.

Print on demand/email on demand

MediaRich can be used as a core component to create a Web-accessible variable content composition and distribution engine that combines integrated page layout, font generation, and image editing capabilities for a “distribute and print” solution. Directly integrate real-time data to segment, create, preview and publish to any email solution or printer.

Syndication simplified

For companies that syndicate their product information through multiple channels and across affiliate, supplier, and partner Websites, MediaRich provides unlimited control over the display of images for all Web sites. Companies can simply match the specific design requirements for the deployment.

Shrink the workflow path

Automate communication processes with corporate-approved media and templates by integrating MediaRich & your Digital Asset Management System. MediaRich loads images, video & media right from your DAM.

Reduce, reuse, repurpose

Dramatically reduce the production and development costs of creating visually rich, compelling content. No need to store multiple versions for Web, print, and mobile. Create your self-service organization.

Images, media and video - meet batch automation

By automating the ”mass customization” process of creating templates for images and videos, customers have dramatically reduced both the time and costs associated with image and video production and distribution.

Say good-bye to old-school workflows

Generate and serve images and videos for Web, print, or device apps. MediaRich® replaces traditional image and video processing methods by automating media processing workflow operations. This significantly reduces costs and dramatically improves scalability.

IRIS MEDIARICH® SERVER

MediaRich Under the Hood

MediaRich's key differentiating features include:

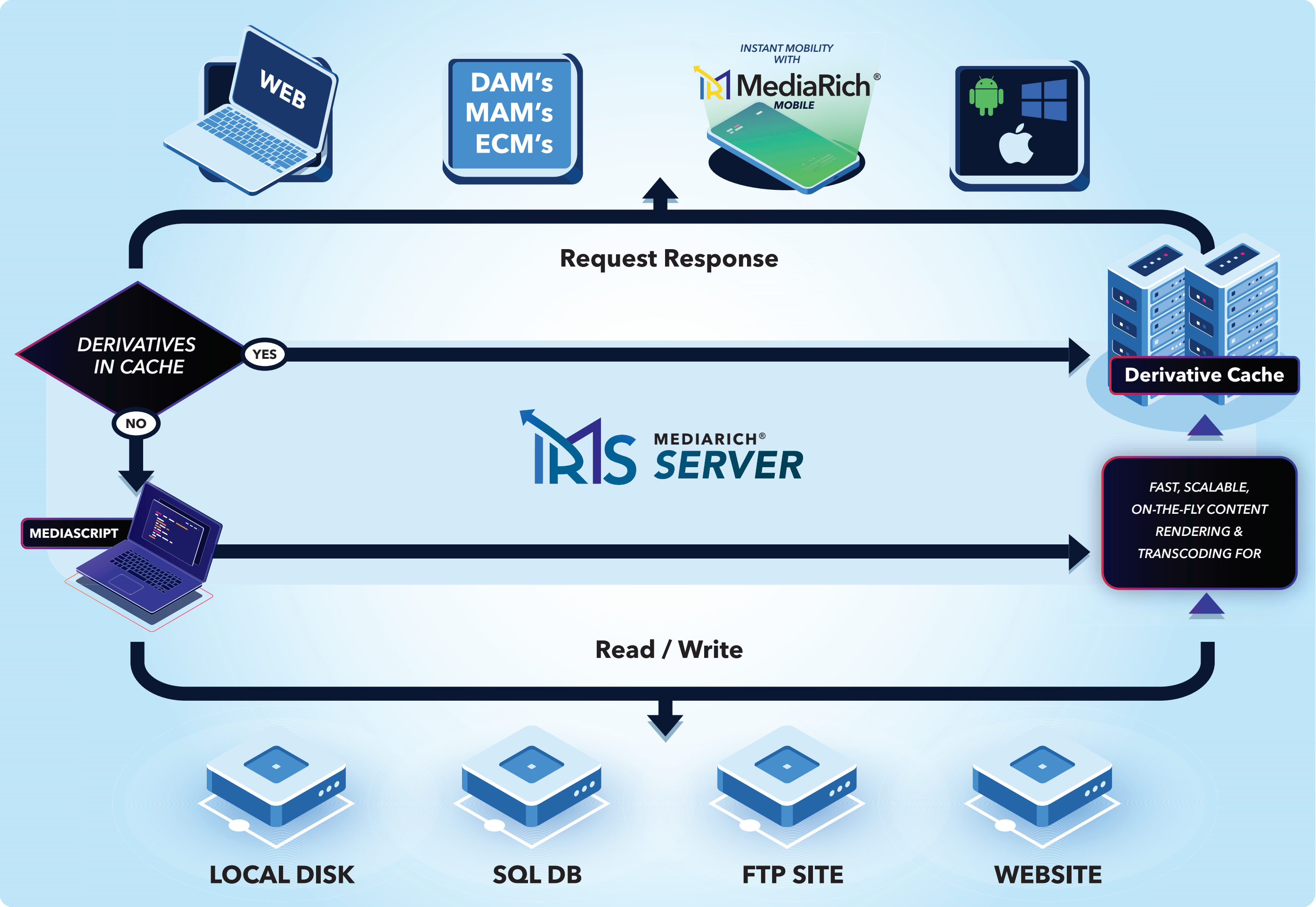
Automatically manages all your renditions and keeps track so you don't have to!

MediaRich includes an intelligent caching mechanism that creates and saves dynamically generated content making it available for subsequent requests. At the first request for a specific asset, MediaRich generates the requested item. Additional requests are served out of the cache with no new processing required and are equivalent to serving the digital content as a static file from a normal Web server. This enables a host of new capabilities such as instant resizing of visuals, low bandwidth high-quality zoom and pan, dynamic experience creation, automatic branded video creation, or wherever your imagination can take you.

MediaScript Language

MediaRich includes an intelligent caching mechanism that creates and saves dynamically generated content making it available for subsequent requests. At the first request for a specific asset, MediaRich generates the requested item. Additional requests are served out of the cache with no new processing required and are equivalent to serving the digital content as a static file from a normal Web server. This enables a host of new capabilities such as instant resizing of visuals, low bandwidth high quality-zoom and pan, dynamic experience creation, automatic branded video creation, or wherever your imagination can take you.

Included is a complete library of support objects such as XML, DOM Parser Generators, File and Directory Assessors, string and regular expression libraries, and many more. MediaRich includes an extensive set of operations for querying and responding to requests for media reading, manipulating and processing sound, video, image, vector, and text content. It also manages the rest of the MediaRich system.



Ingestion and dynamic mobility of content from practically any source!

MediaRich can read almost any file and can write to any storage system locally or remotely. Key auto-assembly and auto "normalization" of content enables on-demand video ingestion and preparation without human intervention or old-school batch processing techniques.

DON'T JUST MANAGE YOUR CONTENT, MEDIARICH IT!

> MediaRich Server with the new A/V CORE had an immediate impact in server resource reduction, speed of processing, next-generation automatic mobility, simplified administration, and setup. So much so, that we decided to take it right to production with a few weeks of testing, during our busiest shopping season and will never look back.

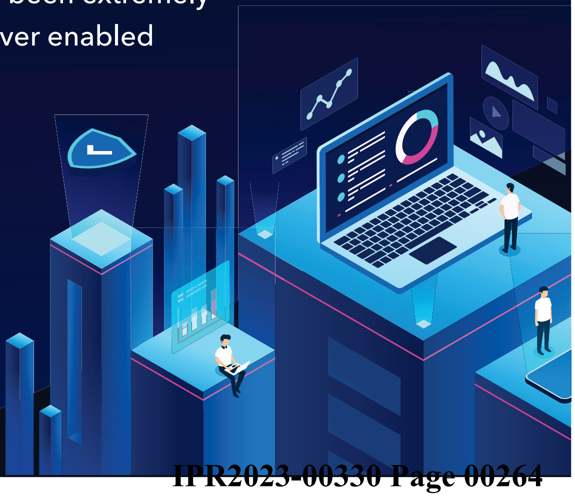
Erik K. Schaeffer
Lead Systems Engineer

> We looked at alternatives for deploying images and video for Cisco.com, but nothing came close to Equilibrium MediaRich Server in features, scalability, or return on investment.

Janet Starkey Wallin
Web Marketing & Strategy

> MediaRich extends our SharePoint platform and provides all the capabilities we needed, from view anywhere technology, MediaRich's seamless visual integration along with SharePoint's version control and publishing libraries, to the MetaBatch processing and HTML5 uploader, we have been extremely happy with the team at Equilibrium and their MediaRich Server enabled products...

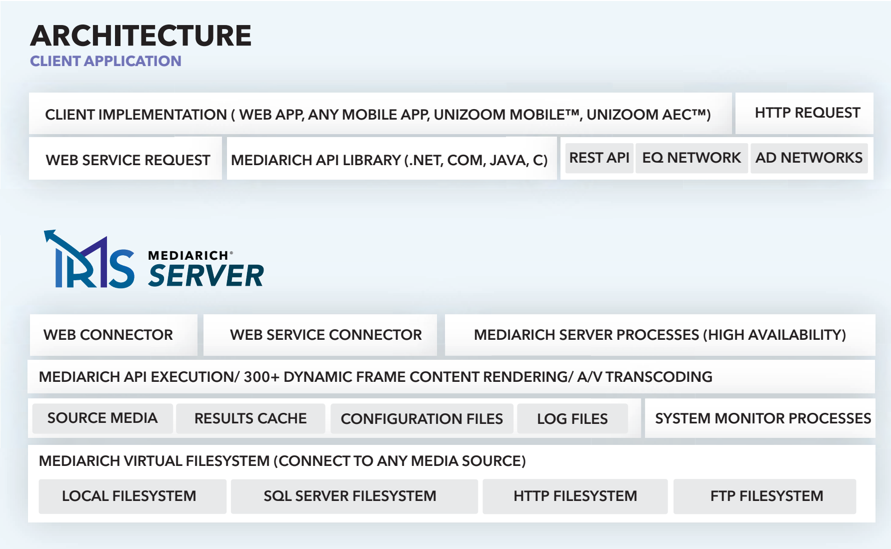
Walter Curtis
Sharepoint AED / SME -U.S. Dept. of Energy Office of Legacy Management



The MediaRich Platform:

End-to-End Enterprise Class Content Preparation

The MediaRich Platform is a patented server-based media processing engine that automates image and video production, enabling direct delivery of media assets to Web pages, digital asset management systems, and virtually any device or app. MediaRich streamlines production workflow, complementing existing technologies while reducing costs and accelerating time-to-market and publication of rich visual content. MediaRich can interface with any system, application, and Web infrastructure.



ARCHITECTURE

CLIENT APPLICATION

CLIENT IMPLEMENTATION (WEB APP, ANY MOBILE APP, UNIZOOM MOBILE™, UNIZOOM AEC™) | HTTP REQUEST

WEB SERVICE REQUEST | MEDIARICH API LIBRARY (.NET, COM, JAVA, C) | REST API | EQ NETWORK | AD NETWORKS

MEDIARICH SERVER

WEB CONNECTOR | WEB SERVICE CONNECTOR | MEDIARICH SERVER PROCESSES (HIGH AVAILABILITY)

MEDIARICH API EXECUTION/ 300+ DYNAMIC FRAME CONTENT RENDERING/ A/V TRANSCODING

SOURCE MEDIA | RESULTS CACHE | CONFIGURATION FILES | LOG FILES | SYSTEM MONITOR PROCESSES

MEDIARICH VIRTUAL FILESYSTEM (CONNECT TO ANY MEDIA SOURCE)

LOCAL FILESYSTEM | SQL SERVER FILESYSTEM | HTTP FILESYSTEM | FTP FILESYSTEM

Customize and integrate through MediaScript, Equilibrium's powerful procedural scripting language, compliant with the EMCA Script standard and specification, the MediaRich Platform provides templating capabilities for video and images that are as easy as text management in content management systems.

Support for 450+ Filetypes

Native Image Filetypes

.PIX, .AVIF, .BMP, .DNG, .CRW, .DCR, .KDC, .K25, .DC2, .ORF, .RAF, .RAW, .RW2, .NEF, .CR2, .MOS, .CS1, .ARW, .3FR, .ERF, .MEF, .MRW, .PEF, .SR2, .SRF, .X3F, .FFF, .RED, .IIQ, .SRW, .STI, .R3D, .ARI, .QTK, .RDC, .HDR, .IA, .EPS, .EPSF, .GIF, .HEIC, .HEIF, .HEIX:, .MIF1, .AI, .INDD, .IND, .JPG, .JPE, .JPEG, .JFIF, .JP2, .JPX, .J2K, .JPF, .JPC, .J2C, .JCC, .PCX, .PDF, .PSD, .PSB, .PCT, .PICT, .PNG, .PS, .TIF, .TIFF, .TF8, .BTF, .WBMP, .WEBP, .SGI, .BW, .RGBA, .TGA, .SWFT, .PPM, .PBM, .PPMB, .PBMA, .PBMB, .RGB, .CMYK

Native Audio/Video Filetypes

.AC3, .ADTS, .ADX, .AIFF, .APNG, .APTX, .APTX_HD, .ASF, .AST, .ASF_STREAM, .AU, .AVI, .AVM2, .CAF, .DAUD, .DNXHD, .DTS, .DV, .EAC3, .F4V, .FITS, .FLAC, .FLV, .G722, .G723_1, .G726, .G726LE, .GXF, .H261, .H263, .H264, .HLS, .IPOD, .ISMV, .LATM, .M4V, .MATROSKA, .MATROSKA, .MJPEG, .MMF, .MOV, .MP2, .MP3, .MP4, .MPEG, .VCD, .MPEG1VIDEO, .DVD, .SVCD, .MPEG2VIDEO, .VOB, .MPEGTS, .MPJPEG, .MXF, .MXF_D10, .NUT, .OGA, .OGG, .OGV, .OPUS, .ALAW, .MULAW, .VIDC, .PSP, .RM, .SBC, .FILM_CPK, .SPDIF, .SWF, .3G2, .3GP, .TRUEHD, .TTA, .W64, .WAV, .WEBM, .WTV

RAW Camera FileTypes

.DNG, .CRW, .DCR, .KDC, .K25, .DC2, .ORF, .RAF, .RAW, .RW2, .NEF, .CR2, .MOS, .CS1, .ARW, .3FR, .ERF, .MEF, .MRW, .PEF, .SR2, .SRF, .X3F, .FFF, .RED, .IIQ, .SRW, .STI, .R3D, .ARI, .QTK, .RDC, .HDR, .IA

Document, Drawing, and Presentation Filetype Support

.123, .CSV, .ODS, .OTS, .PXL, .XLS, .XLSX, .XLSM, .XLM, .XLC, .XLW, .XLSB, .XLTM, .XLT, .XLTX, .WK1, .WKS, .SLK, .STC, .SXC, .UOF, .UOS, .WB2, .DOC, .DOCX, .DOCM, .RTF, .HTML, .HTM, .OTH, .ODT, .OTT, .DOT, .DOTM, .DOTX, .602, .HWP, .ODM, .PDB, .STW, .SDW, .SGL, .SXG, .SXW, .WPD, .XML, .WBK, .ODG, .OTG, .PCD, .PGM, .RAS, .XBM, .XPM, .DXF, .EMF, .MET, .SGF, .SGV, .SVG, .WMF, .SXD, .STD, .THM, .PUB, .VSD, .VDX, .VSDX, .PMD, .PPT, .PPS, .PPSX, .PPTX, .PPTM, .ODP, .OTP, .CGM, .POT, .POTM, .POTX, .SDP, .SXI, .UOP

MINIMUM SYSTEM REQUIREMENTS

Minimum recommended configuration for imaging, documents, office applications (no heavy video transcoding, 4k or UHD video transcoding):

- Dual 3+ GHz Intel® Xeon Sandy Bridge and later
- 8 GB Ram
- Microsoft Windows Server 2012R2 and later
- Microsoft Internet Information Services® (IIS) and ASP.NET with Sub-components

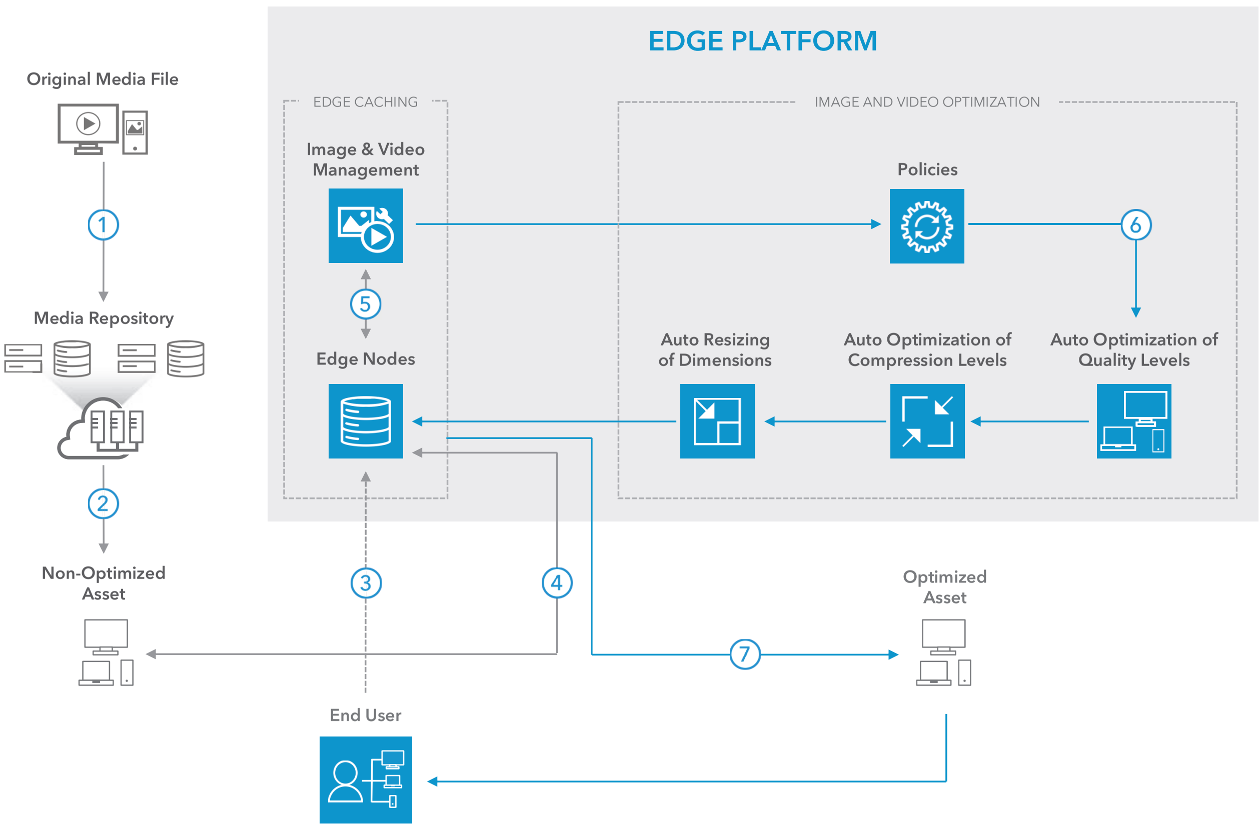
Minimum recommended configuration when processing large HQ videos such as 4k, UHD or 1080P, images, multi-page documents (Bare metal is highly recommended for video transcoding!):

- 8 Core Intel®Xeon Sandy Bridge and later
- Dedicated MediaRich Server 8 Core License in shared cache configuration
- Dedicated cache volume 4 x 250 GB Disk 15K SAS RAID-10
- OS-Volume 2x150 GB/SAS/10K mirror
- 32 GB Ram

Exhibit H

IMAGE AND VIDEO OPTIMIZATION

Reference Architecture



OVERVIEW

Image & Video Manager is a software as a service (SaaS) solution that automatically optimizes and enhances images and videos for every user on the fly. It provides each user the ideal combination of quality, format, and size that is best suited for that user's browser, device, and network connection at the very moment they access a website or mobile app.

Image & Video Manager makes responsive web design significantly easier, cheaper, and more efficient.

1 Creative services, third parties, or users provide image or video asset in the form of one high-quality, pristine file. Only this one file needs to be stored on the origin (CMS, DAM, HTTP) server.

2 Image or video file is embedded in user experiences of applications, web pages, or apps. No optimization code and only one original file is needed.

3 The Akamai Intelligent Edge Platform receives a user request and obtains detailed information about the user context, including browser, device types, and the quality level of the end user's current networking connection.

4 A copy of the original is cached on Akamai's distributed edge network.

5 Image & Video Manager works on top of the Akamai Intelligent Edge Platform, and utilizes its edge compute capabilities and user context insights to optimize image and video assets.

6 Optimizations can be defined and controlled via policies, and can be applied to individual images (or groups of images) based on rules.

7 The Akamai Intelligent Edge Platform delivers great user experiences and optimizes infrastructure offload as it scales to meet peak traffic demand.

KEY PRODUCTS

Image and/or Video Optimization ▶ Akamai Image & Video Manager

Edge Caching / Content Delivery Network ▶ Akamai Ion

Published 10/20

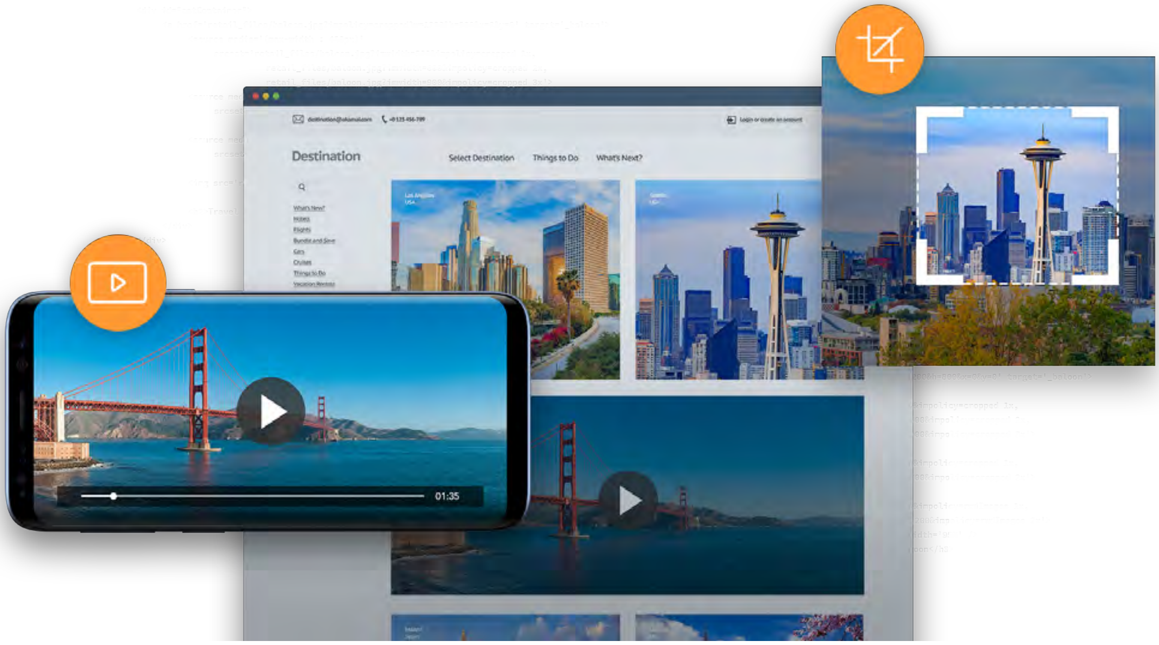
Akamai

Exhibit I

Akamai DEVELOPER

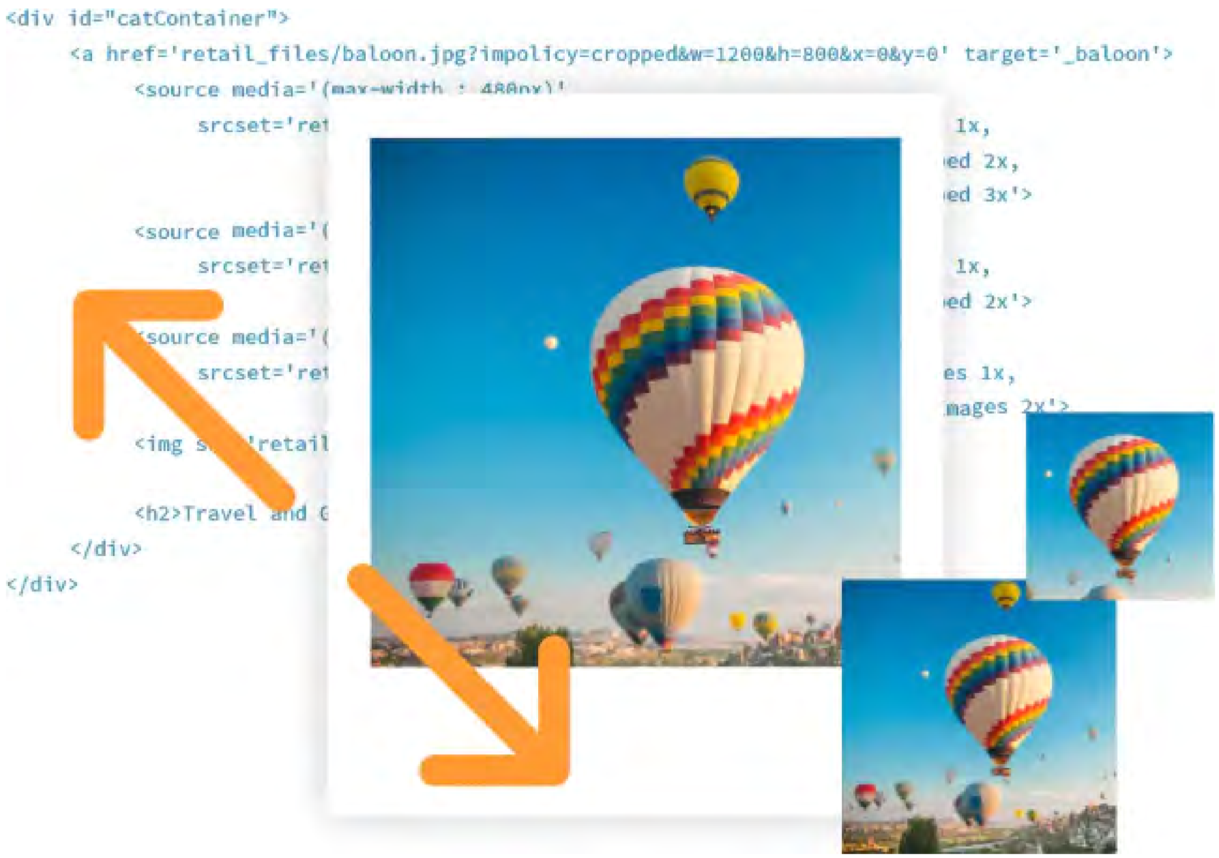
Akamai Image and Video Manager

FREE DEVELOPER TRIAL



Get automated optimizations for images, videos, and animated GIFs.

Akamai Image and Video Manager comes out of the box ready to help developers optimize images, videos, and animated GIFs to deliver great web experiences. Specifically, here are five Image and Video Manager capabilities that are especially popular with developers:



```
<div id="catContainer">
     <a href='retail_files/baloon.jpg?impolicy=cropped&w=1200&h=800&x=0&y=0' target='_baloon'>
          <source media='(max-width : 480px)'
               srcset='ret                                             1x,
                                                                      ed 2x,
                                                                      ed 3x'>

          <source media='(
               srcset='ret                                             1x,
                                                                      ed 2x'>

          <source media='(
               srcset='ret                                            es 1x,
                                                                      mages 2x'>

          <img s  'retail

          <h2>Travel and (
     </div>
</div>
```

Get creative with images via code

With Image and Video Manager, you can chain together transformations on-the-fly with Query String Parameters (QSPs) and be as creative as you want to be with images using code.

Easily crop images , turn on grayscale, apply watermarks and more, all at the code level.

VIEW DOCUMENTATION

CLICK ANY URL BELOW TO SEE THE TRANSFORMATIONS IN ACTION.

Resize and maintain aspect ratio

https://developer.akamai.com/image-manager/img/gallery-1.jpg?impolicy=resize&width=500

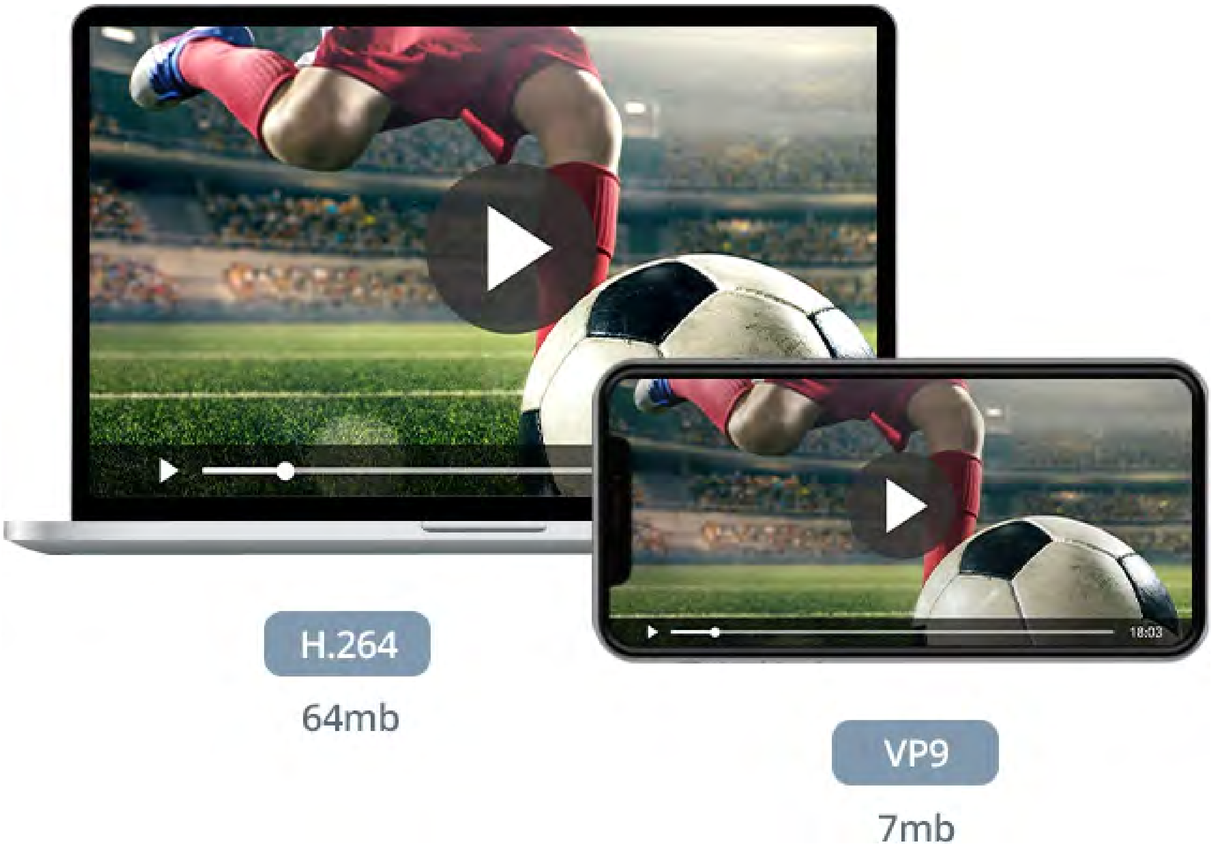
Resize using manual hints

https://developer.akamai.com/image-manager/img/gallery-1.jpg?imwidth=500

Dynamic crop

https://developer.akamai.com/image-manager/img/gallery-1.jpg?impolicy=crop&width=1000&height=1000&y=500&x=1800

Auto-crop to face and render in black and white
https://developer.akamai.com/image-manager/img/gallery-1.jpg?impolicy=filter

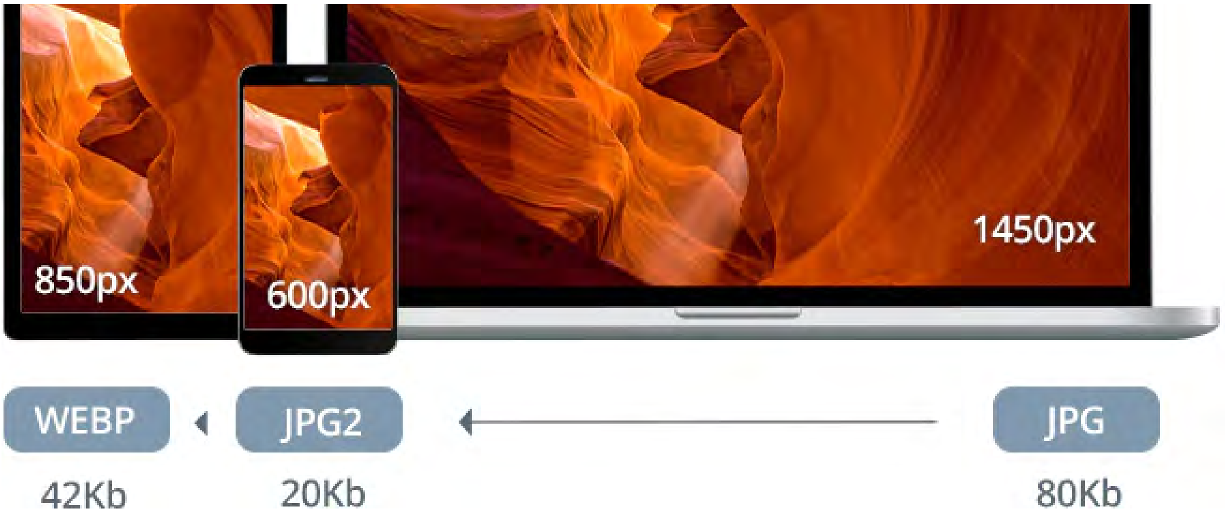


Optimize and deliver your videos

Video content can greatly enhance your user's experience, but it also can cause complications with the performance of your website. Image and Video Manager allows you to deliver high-quality, short-form video without worrying about byte size or bloated page or app weight.

Just save your video (up to five minutes in length) at origin and then let Image and Video Manager do the work, automatically creating and delivering a derivative video with the correct bit-rates, resolutions, and formats based on the user's device.

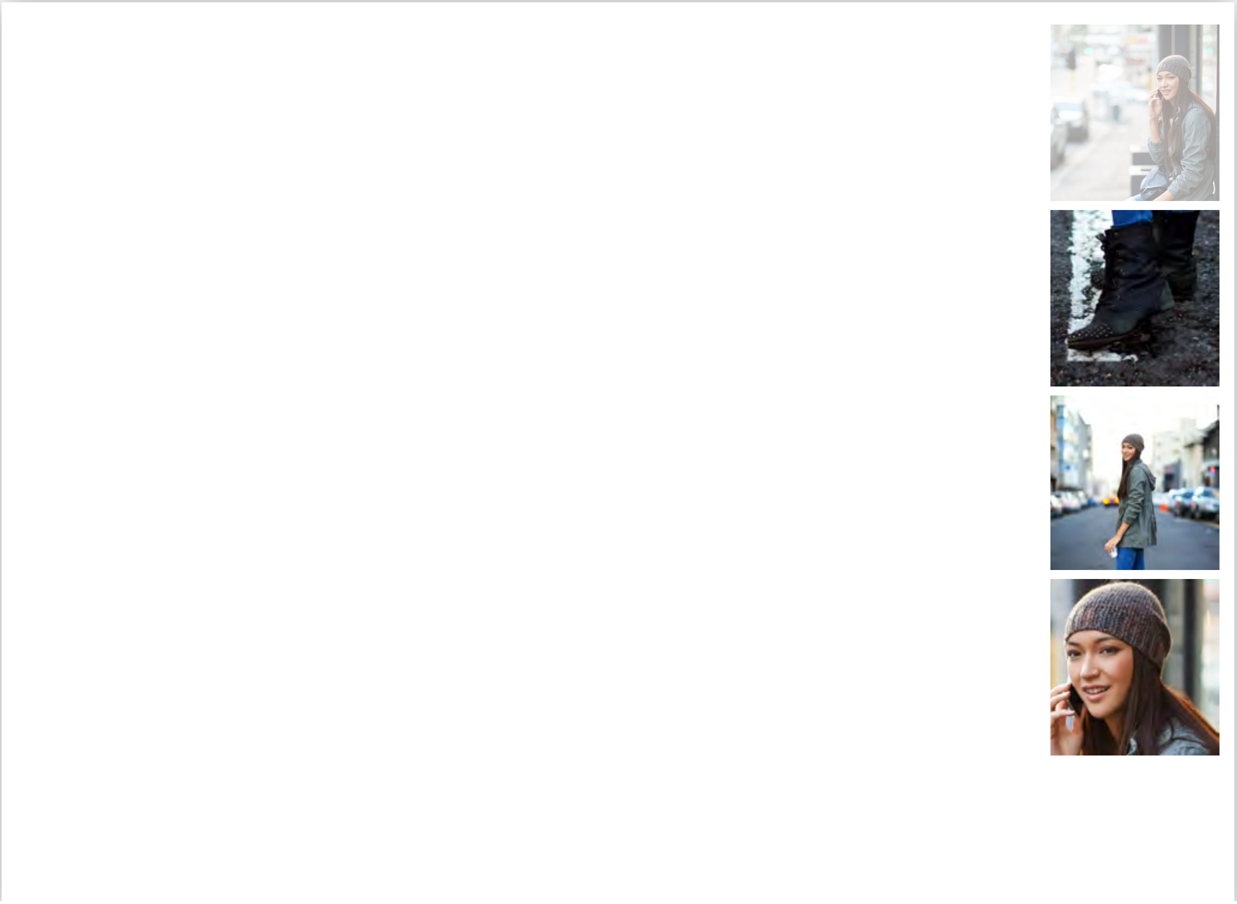
LEARN MORE WATCH THE WEBINAR



Simplify the process of responsive web design (RWD)

The innate complexities associated with RWD can make it a hassle to write logic that will send out the right image to the right end user. Image Manager solves this problem by automating optimal image delivery (e.g., auto-converting formats based on device, browser and/or OS) so that developers can focus on more critical tasks.

ROLL OVER THE IMAGE TO SEE IMAGE MANAGER MAGNIFICATION IN ACTION.



VIEW SOURCE

Easy-to-implement media viewer

With Image Manager's newly released media viewer, developers get a simplified viewer implementation that means one less hassle.

In addition, images are automatically optimized within the viewer, and it's easily customizable so you can fine-tune it for your needs, further streamlining life for developers.

GET STARTED

Original 1.3MB JPEG

Optimized 609KB WEBP

686K
BYTES SAVED

52%
REDUCTION

Exclusive perceptual-quality algorithm

Image Manager includes a perceptual-quality algorithm that means developers no longer need to worry about setting image-level quality.

By automatically optimizing images for maximum perceived quality with minimum image weight, the algorithm lets developers relax knowing that images will be delivered at a specified quality that's optimal for end users.

VALIDATE WITH PIEZ

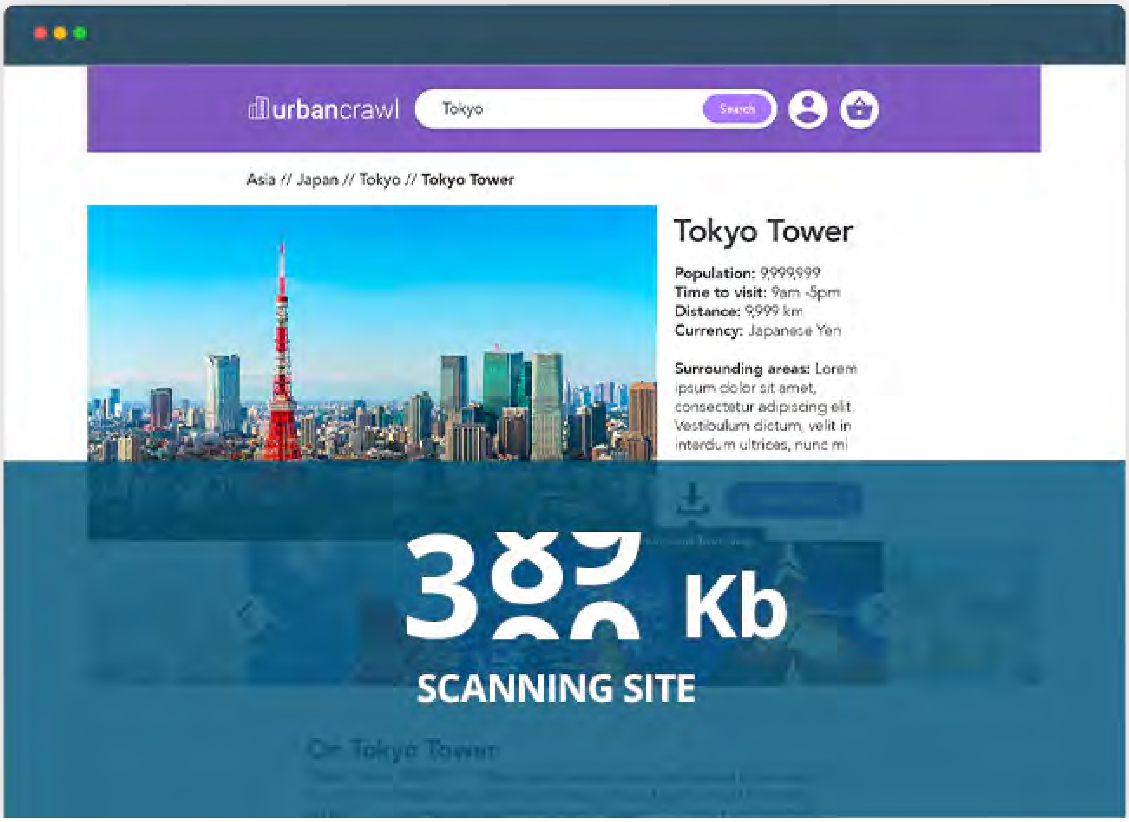
Want more details?

Docs
GO

Cheat Sheet
GO

Cookbook
GO

Webinar
WATCH



Asia // Japan // Tokyo // Tokyo Tower

Tokyo Tower

Population: 9,999,999
Time to visit: 9am -5pm
Distance: 9,999 km
Currency: Japanese Yen

Surrounding areas: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum dictum, velit in interdum ultrices, nunc mi

Put your site to the test.

We offer a free image weight report that helps developers pinpoint on their websites exactly where the weight is and what can be done to optimize it.

GET YOUR FREE REPORT

COMPANY

Akamai.com

Akamai Locations

Contact Us

DEVELOPER

Documentation

Blog

Release Notes

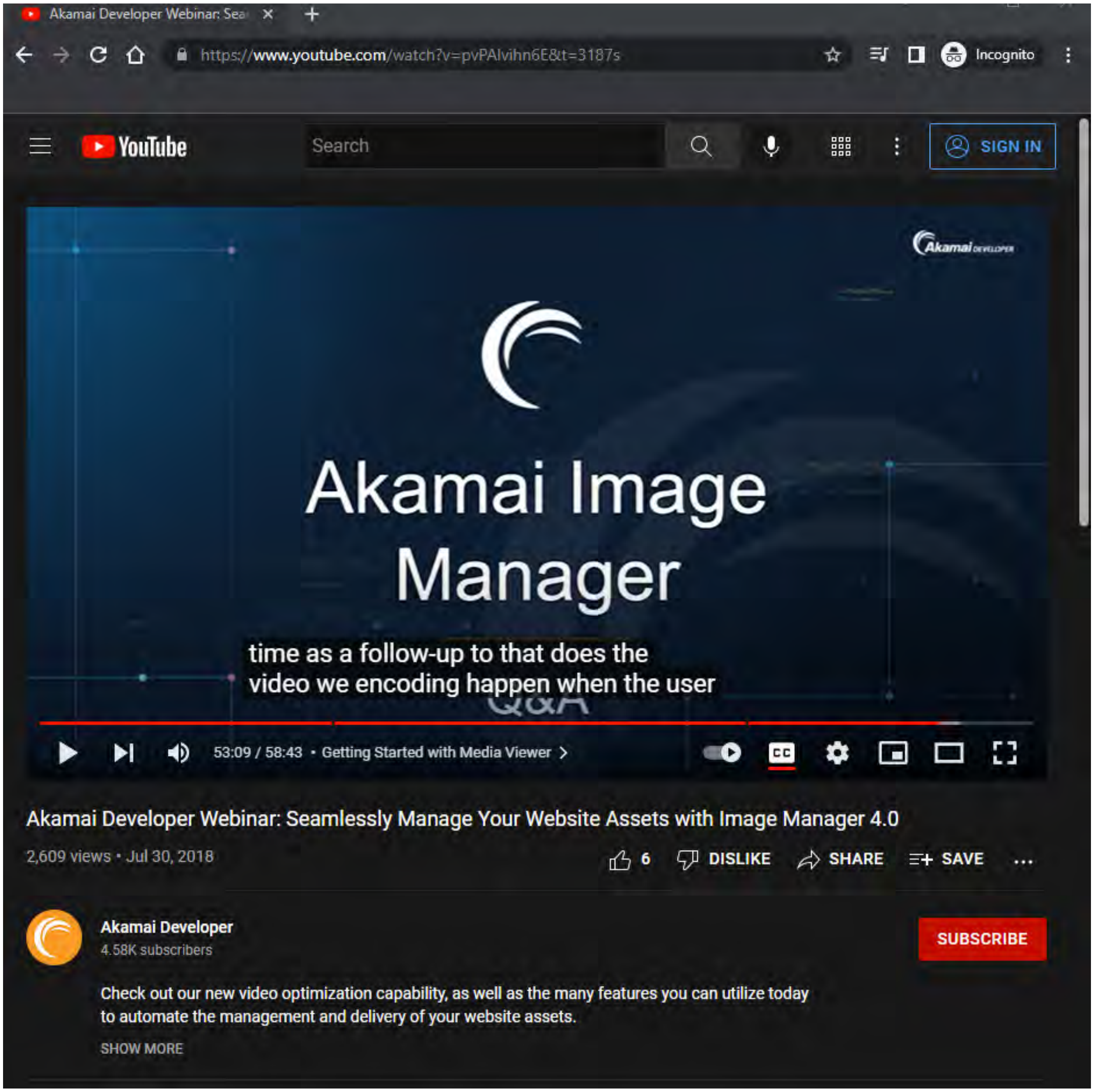
STAY IN TOUCH

SELECT LANGUAGE ▼

Exhibit J

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

(“Webinar – Image Manager 4.0”)


Akamai Developer Webinar: Sea
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
Q&A
53:09 / 58:43 • Getting Started with Media Viewer
6
DISLIKE
SHARE
SAVE
SUBSCRIBE
SHOW MORE

time as a follow-up to that does the video we encoding happen when the user

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

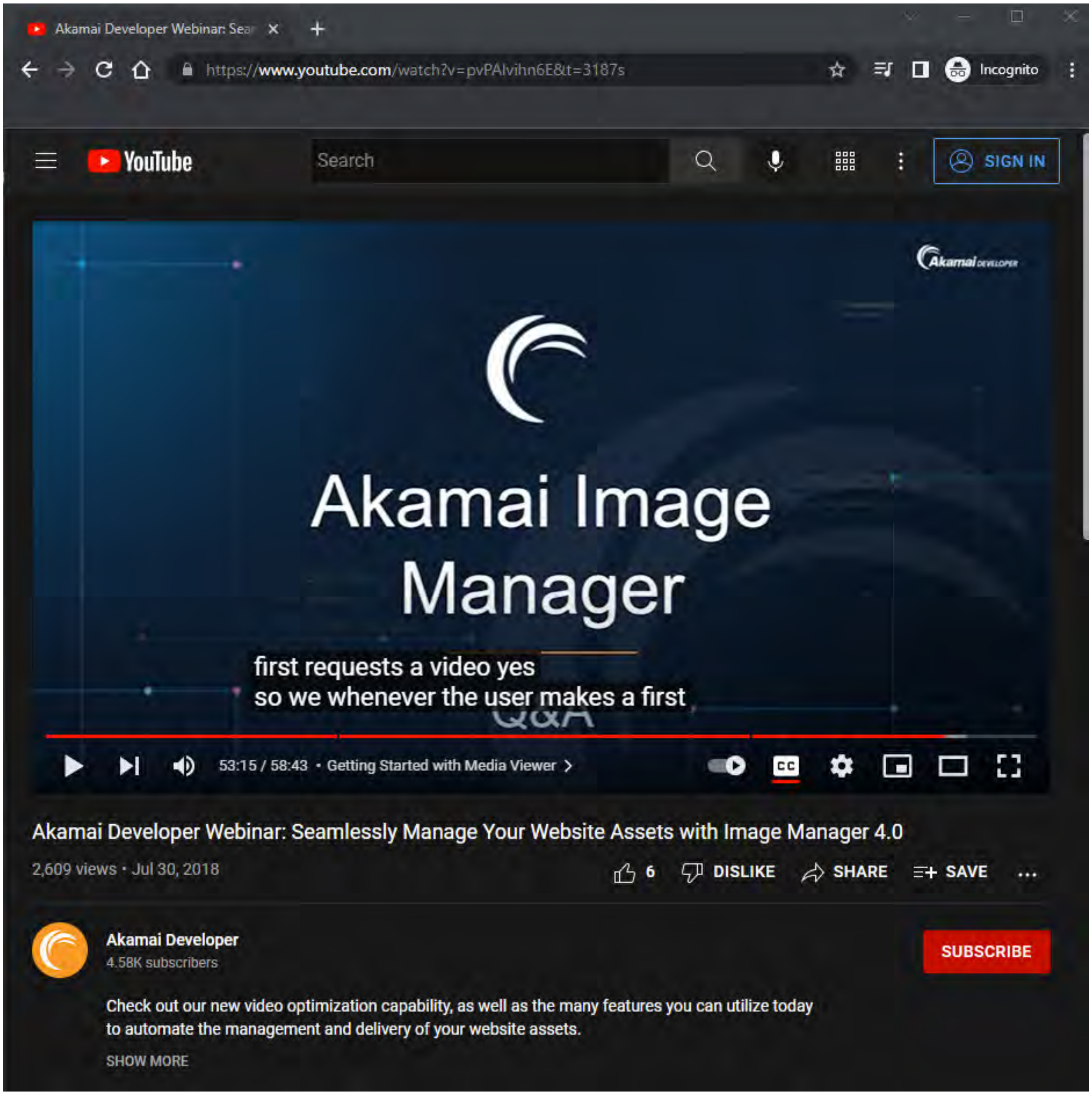
Akamai Developer
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

("Webinar – Image Manager 4.0")



Akamai Developer Webinar: Sear
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
first requests a video yes
so we whenever the user makes a first
53:15 / 58:43 • Getting Started with Media Viewer
6
DISLIKE
SHARE
SAVE
SUBSCRIBE
SHOW MORE

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

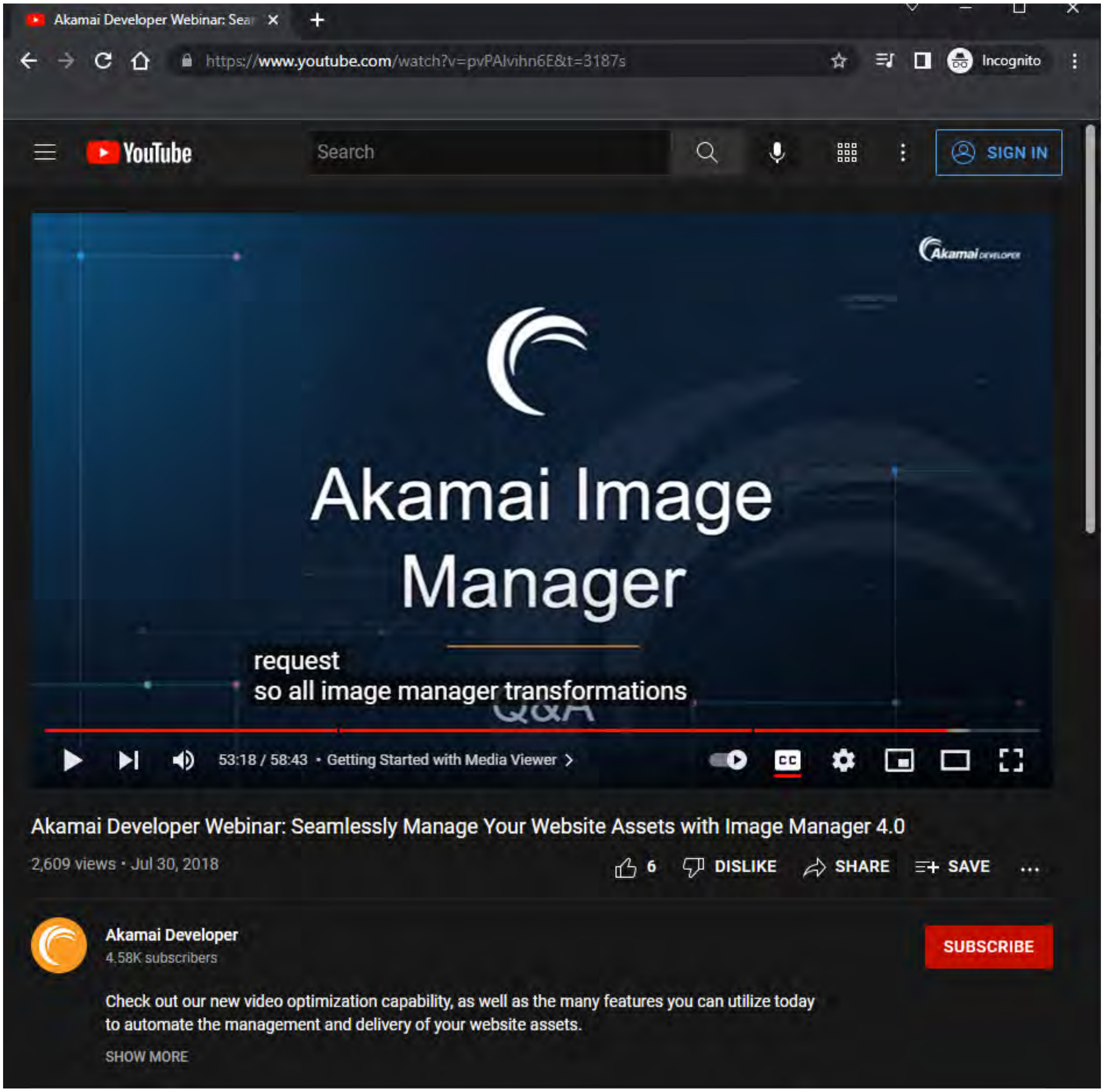
Akamai Developer
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

("Webinar – Image Manager 4.0")


Akamai Developer Webinar: Sear
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
request
so all image manager transformations
Q&A
53:18 / 58:43 • Getting Started with Media Viewer
6
DISLIKE
SHARE
SAVE
SUBSCRIBE
SHOW MORE

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

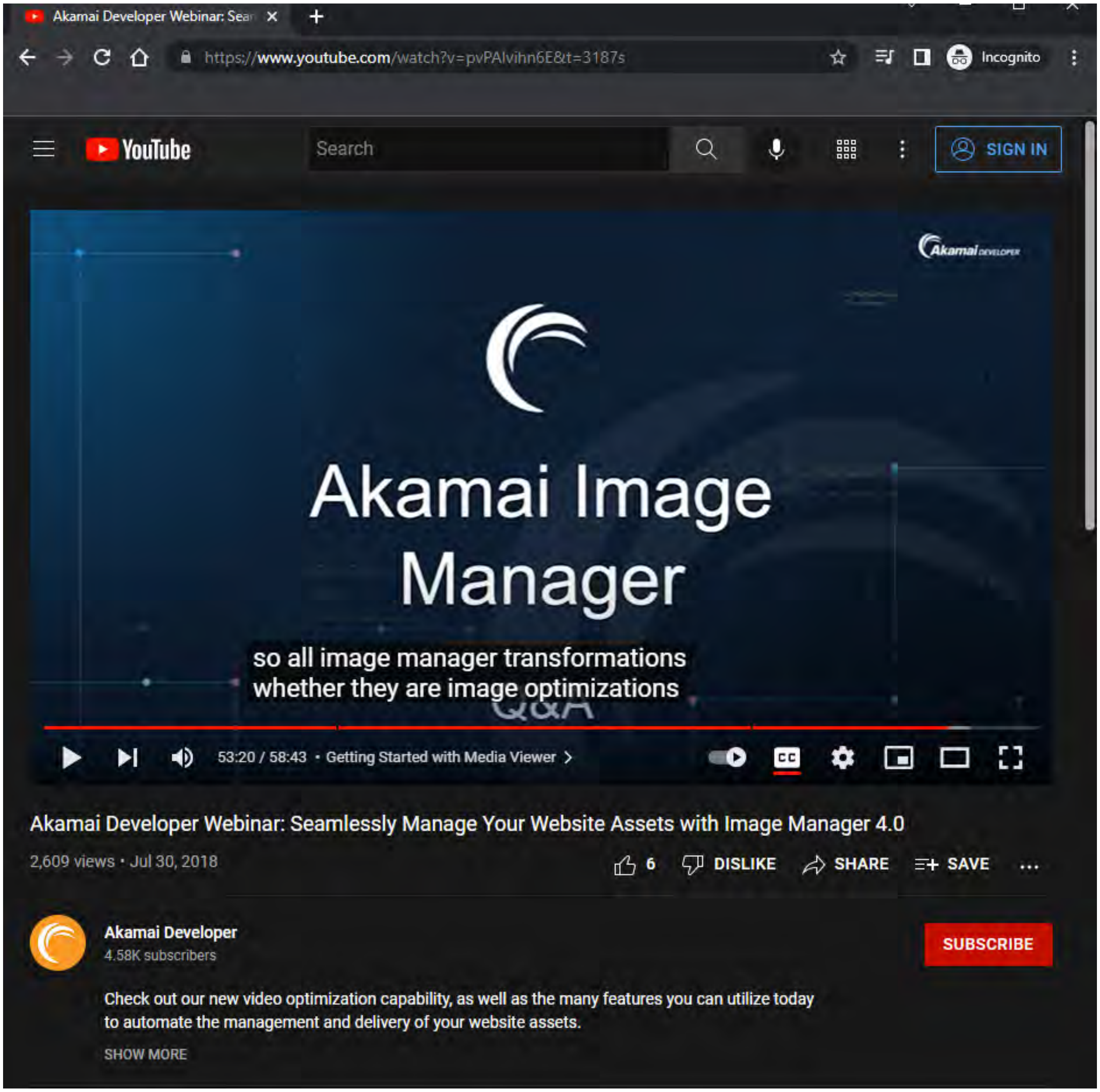
Akamai Developer
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

("Webinar – Image Manager 4.0")


Akamai Developer Webinar: Sean
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
so all image manager transformations
whether they are image optimizations
Q&A
53:20 / 58:43 • Getting Started with Media Viewer

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

6
DISLIKE
SHARE
SAVE
SUBSCRIBE

Akamai Developer
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

SHOW MORE

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

(“Webinar – Image Manager 4.0”)

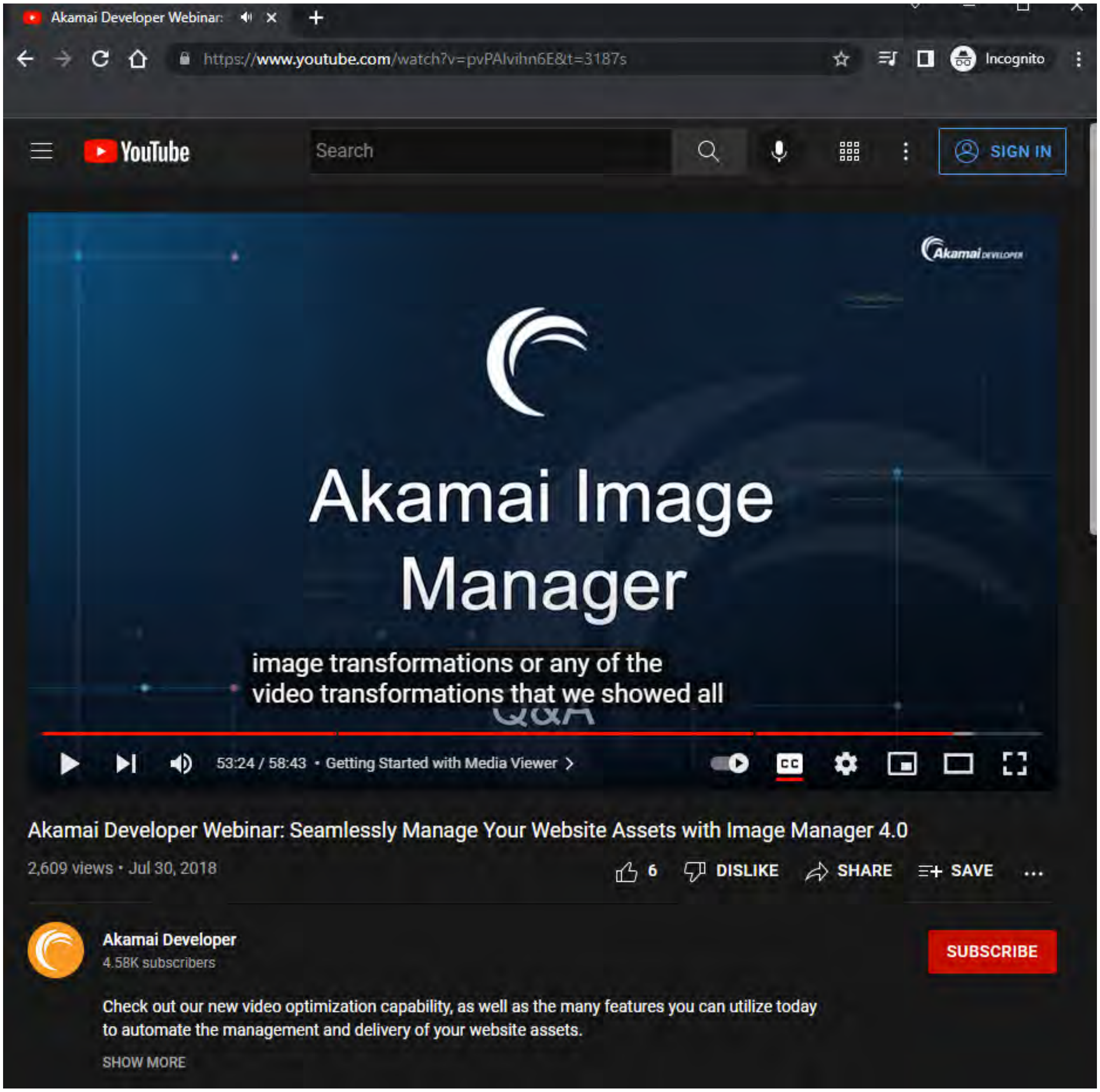

Akamai Developer Webinar:
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
Q&A
53:24 / 58:43 • Getting Started with Media Viewer
6
DISLIKE
SHARE
SAVE
SUBSCRIBE
SHOW MORE

image transformations or any of the video transformations that we showed all

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

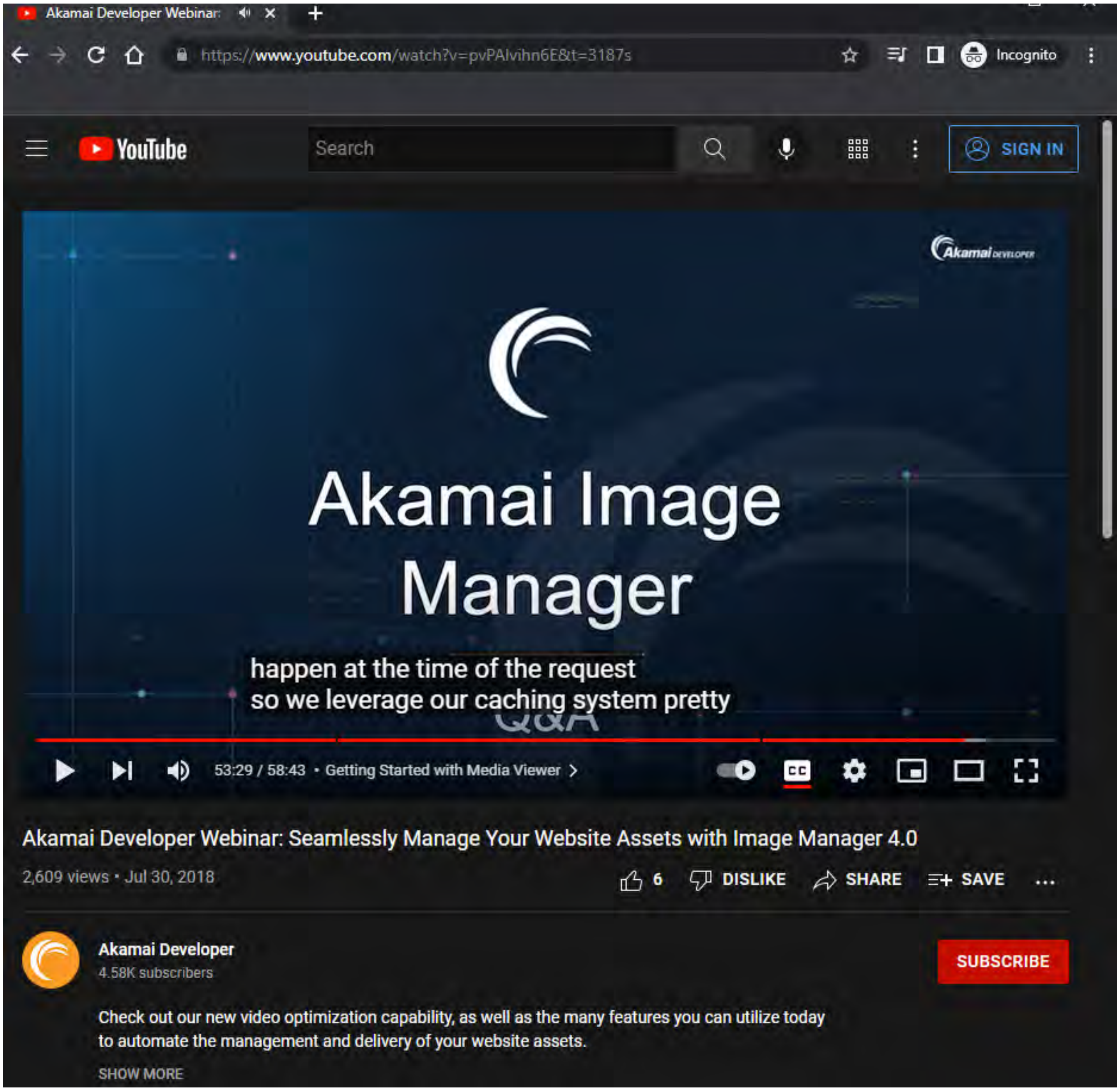
Akamai Developer
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

("Webinar – Image Manager 4.0")

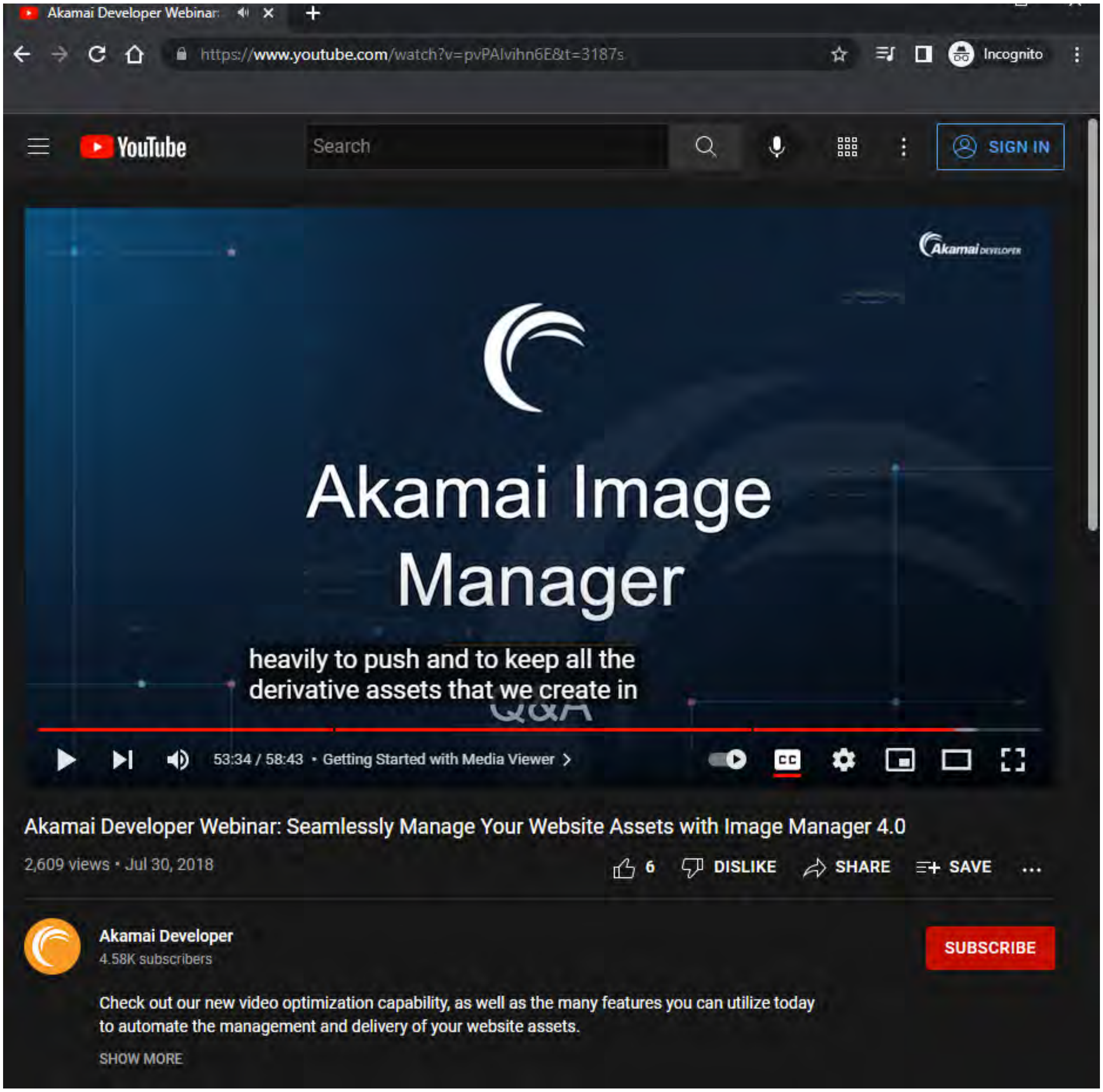

Akamai Developer Webinar:
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
happen at the time of the request
so we leverage our caching system pretty
Q&A
53:29 / 58:43 • Getting Started with Media Viewer
Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0
2,609 views • Jul 30, 2018
6
DISLIKE
SHARE
SAVE
Akamai Developer
4.58K subscribers
SUBSCRIBE
SHOW MORE

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

("Webinar – Image Manager 4.0")


Akamai Developer Webinar:
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
heavily to push and to keep all the
derivative assets that we create in
Q&A
53:34 / 58:43 • Getting Started with Media Viewer
6
DISLIKE
SHARE
SAVE
SUBSCRIBE
SHOW MORE

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

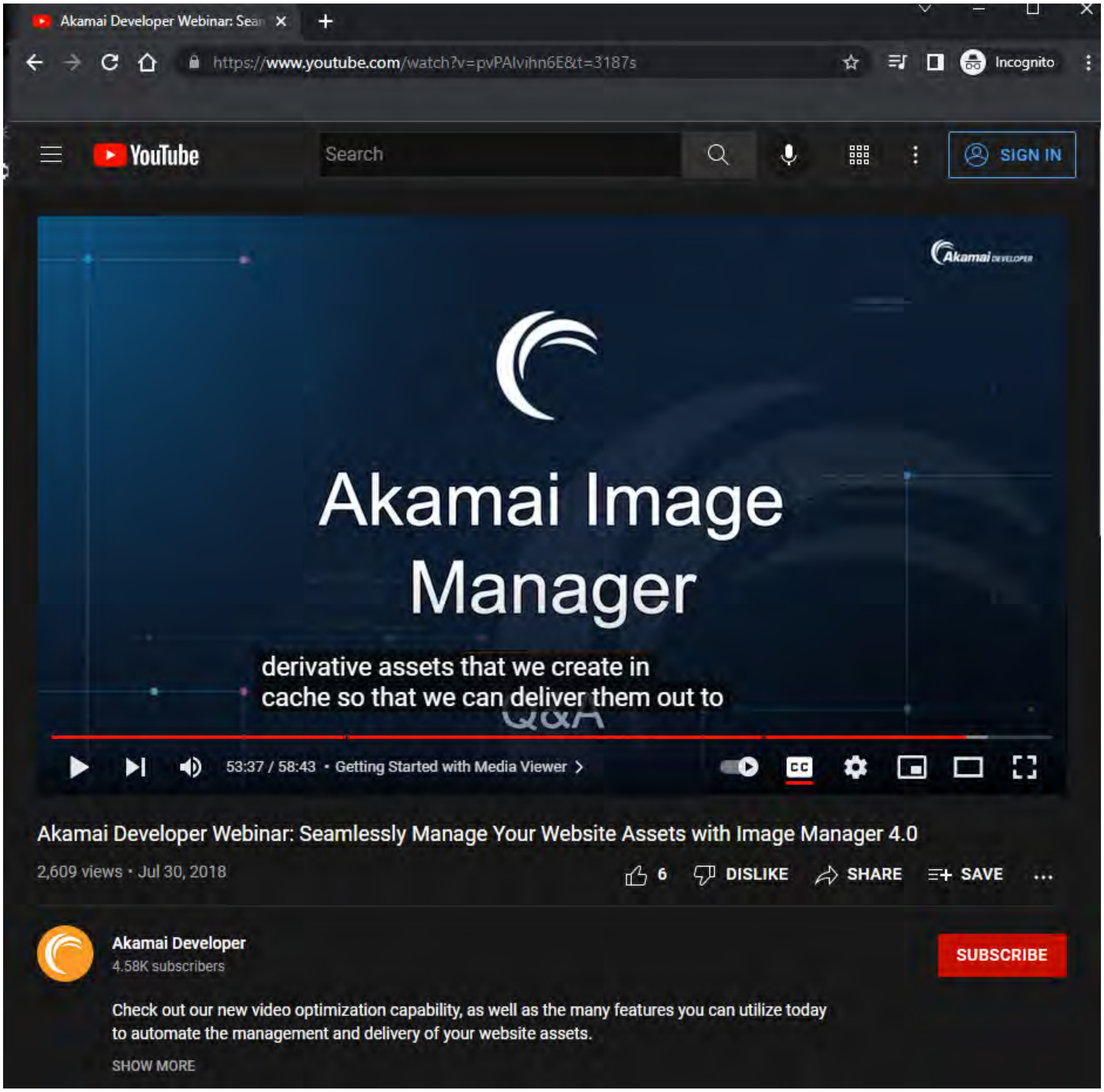
Akamai Developer
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

(“Webinar – Image Manager 4.0”)


Akamai Developer Webinar: Sean
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
derivative assets that we create in
cache so that we can deliver them out to
Q&A
53:37 / 58:43 • Getting Started with Media Viewer
6
DISLIKE
SHARE
SAVE
SUBSCRIBE
SHOW MORE

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

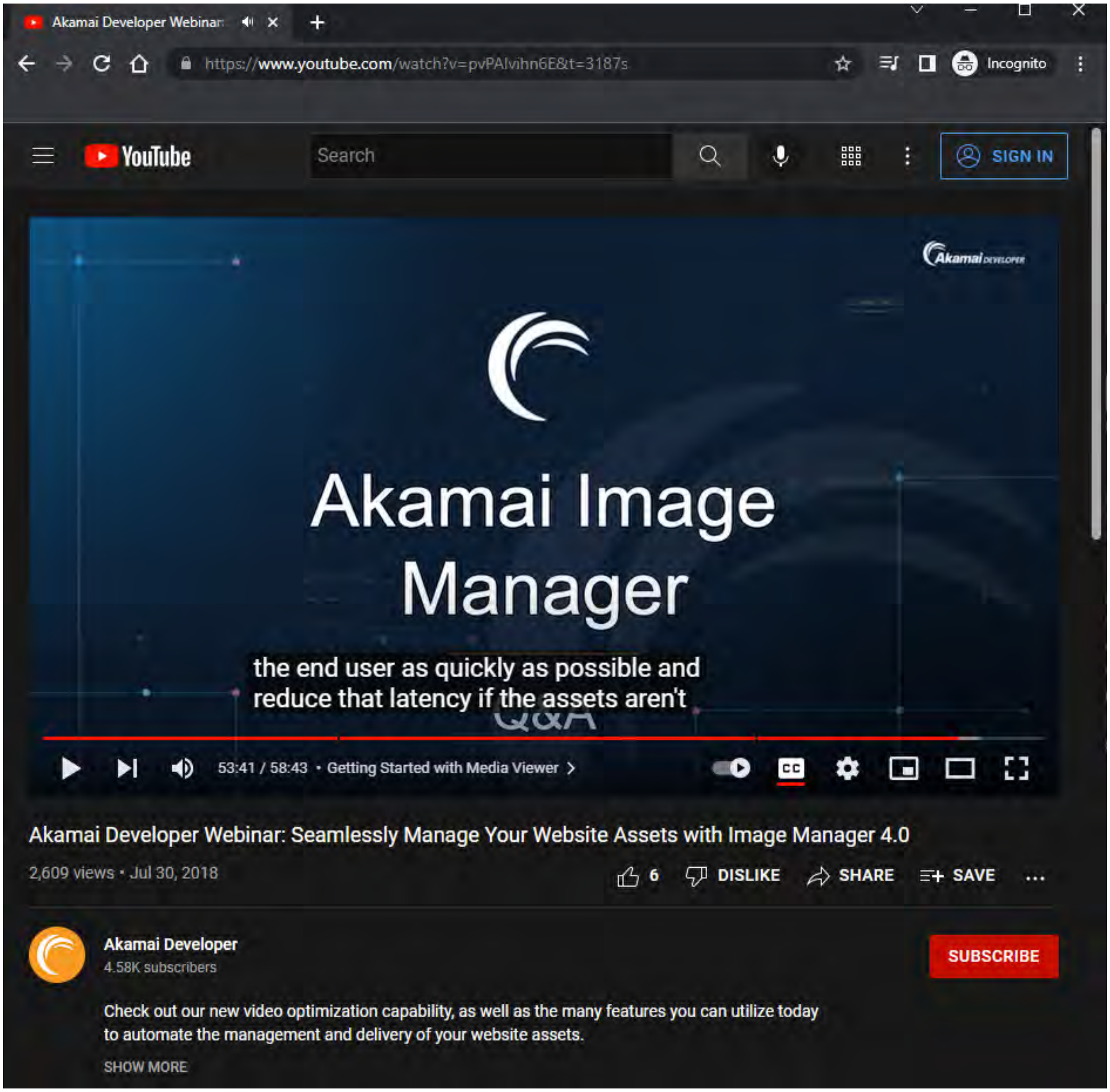
Akamai Developer
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

(“Webinar – Image Manager 4.0”)



the end user as quickly as possible and reduce that latency if the assets aren't

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

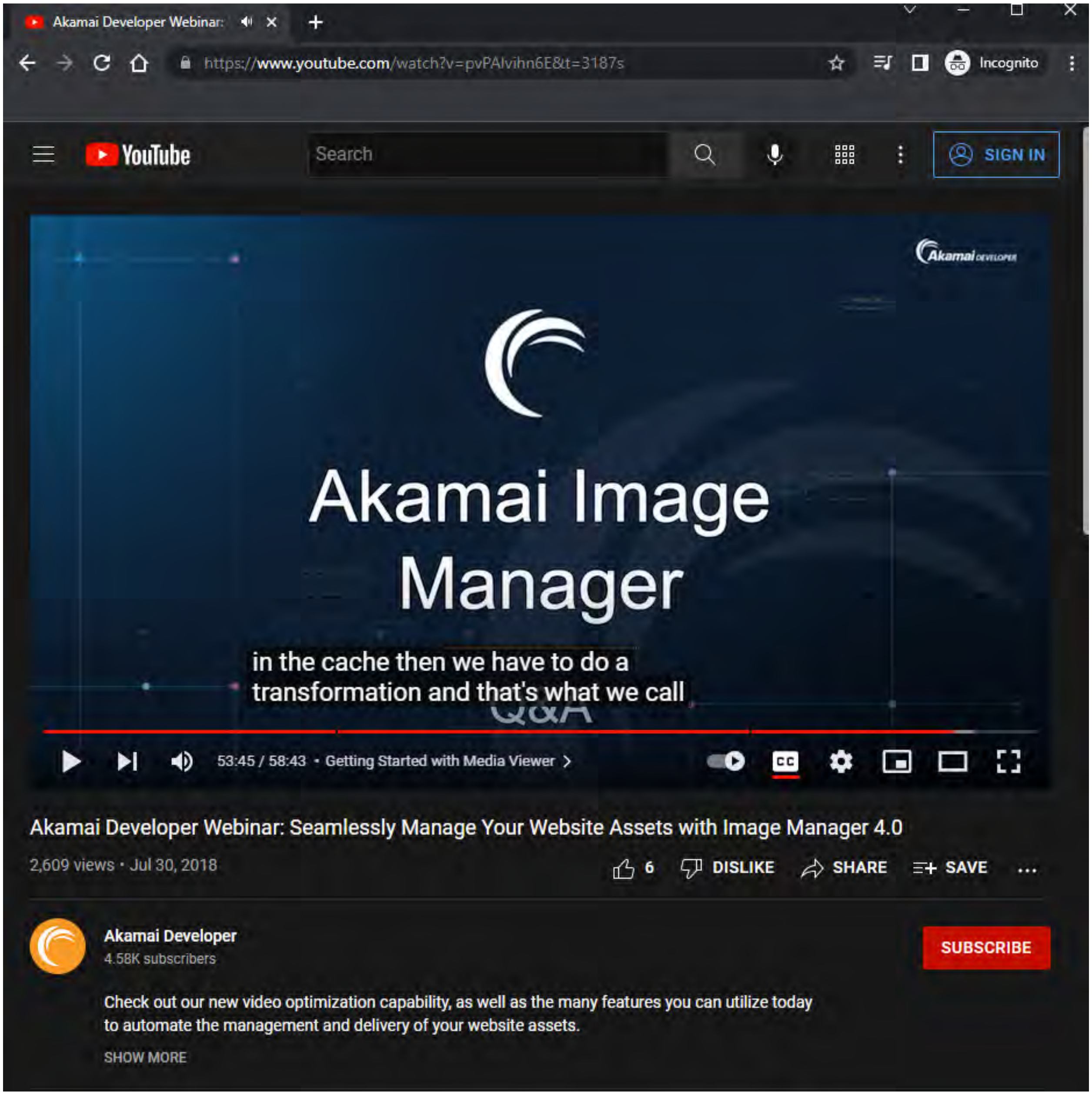
Akamai Developer
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

("Webinar – Image Manager 4.0")


Akamai Developer Webinar:
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
Q&A
53:45 / 58:43 • Getting Started with Media Viewer
6
DISLIKE
SHARE
SAVE
SUBSCRIBE
SHOW MORE

in the cache then we have to do a transformation and that's what we call

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

Akamai Developer

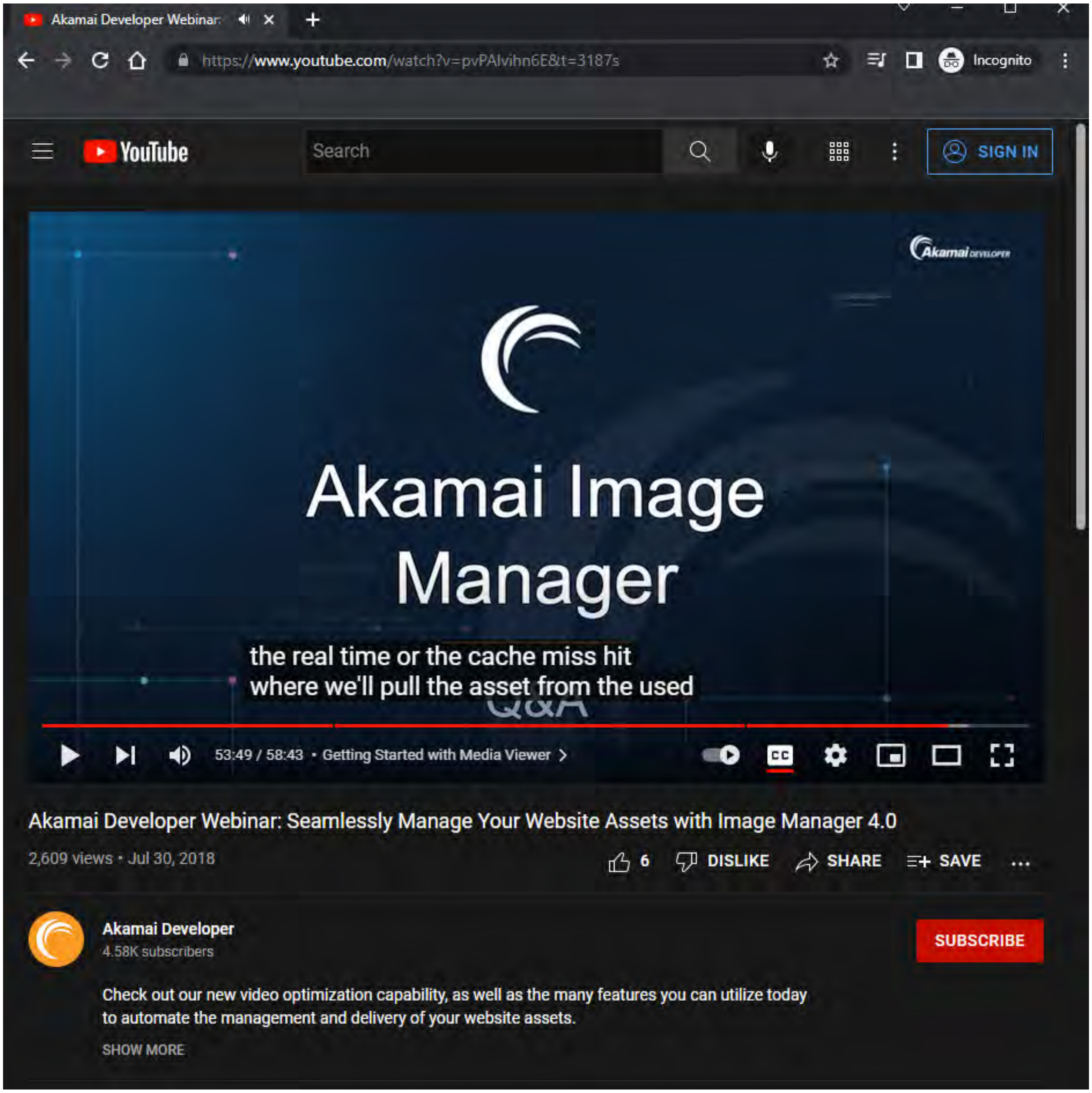
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

("Webinar – Image Manager 4.0")


Akamai Developer Webinar:
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
the real time or the cache miss hit
where we'll pull the asset from the used
Q&A
53:49 / 58:43 • Getting Started with Media Viewer
6
DISLIKE
SHARE
SAVE
SUBSCRIBE
SHOW MORE

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

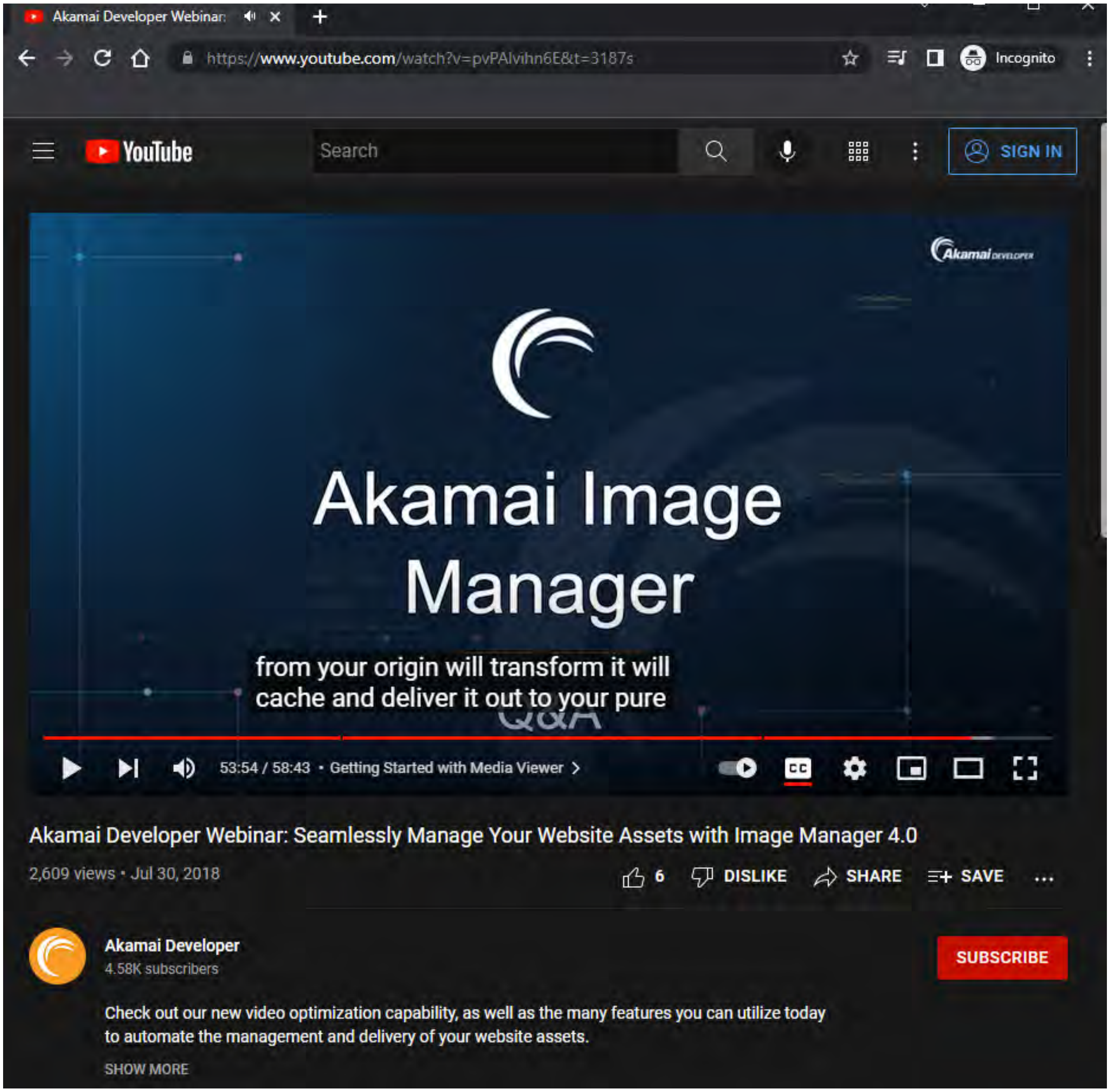
Akamai Developer
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

("Webinar – Image Manager 4.0")



from your origin will transform it will cache and deliver it out to your pure

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

Akamai Developer

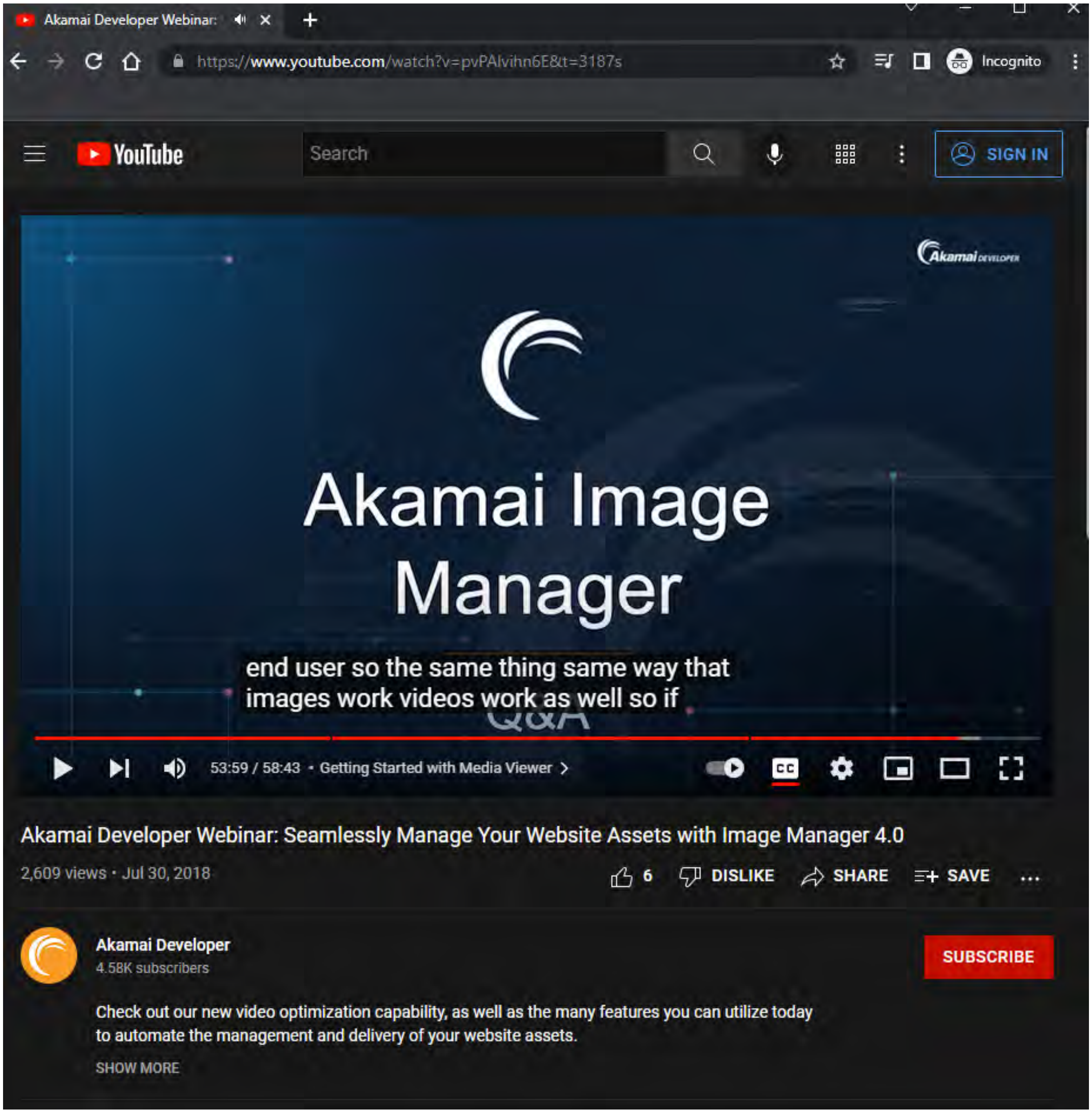
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

("Webinar – Image Manager 4.0")


Akamai Developer Webinar:
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
Q&A
53:59 / 58:43 • Getting Started with Media Viewer
6
DISLIKE
SHARE
SAVE
SUBSCRIBE
SHOW MORE

end user so the same thing same way that images work videos work as well so if

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

Akamai Developer

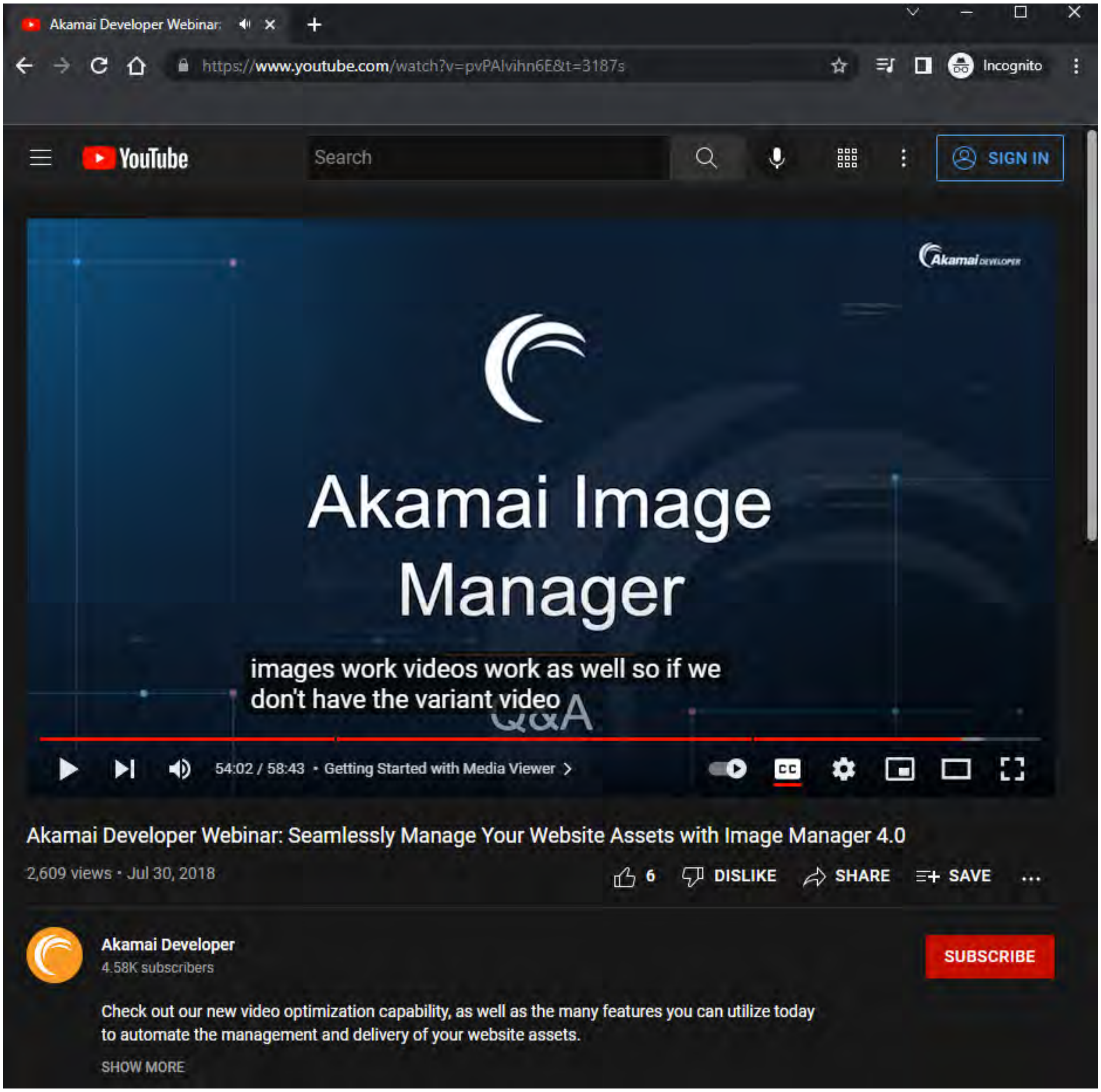
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

(“Webinar – Image Manager 4.0”)



Akamai Developer Webinar:
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
Q&A
54:02 / 58:43 • Getting Started with Media Viewer

images work videos work as well so if we don't have the variant video

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

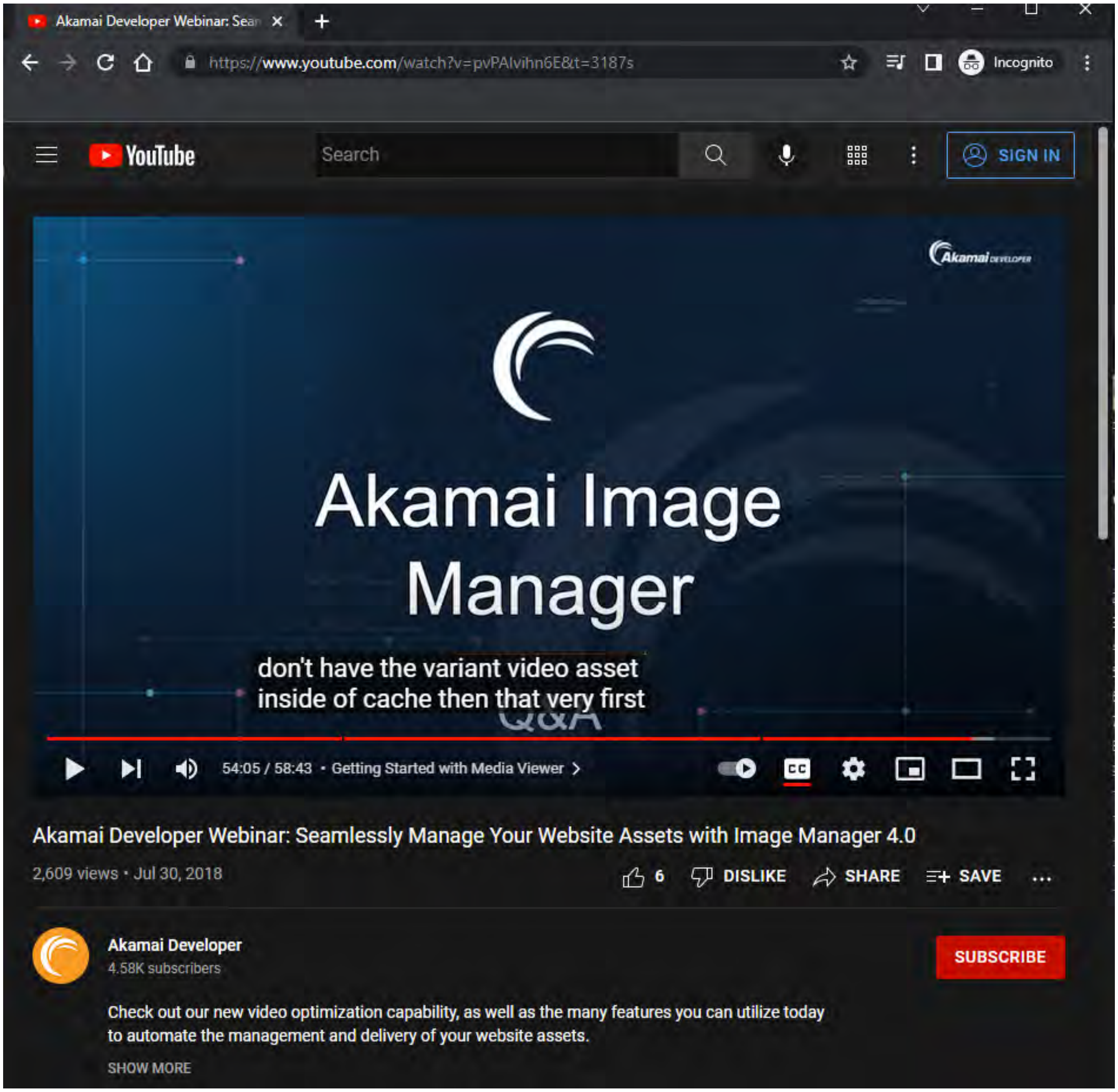
2,609 views • Jul 30, 2018

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

("Webinar – Image Manager 4.0")


Akamai Developer Webinar: Sean
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
don't have the variant video asset
inside of cache then that very first
Q&A
54:05 / 58:43 • Getting Started with Media Viewer

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

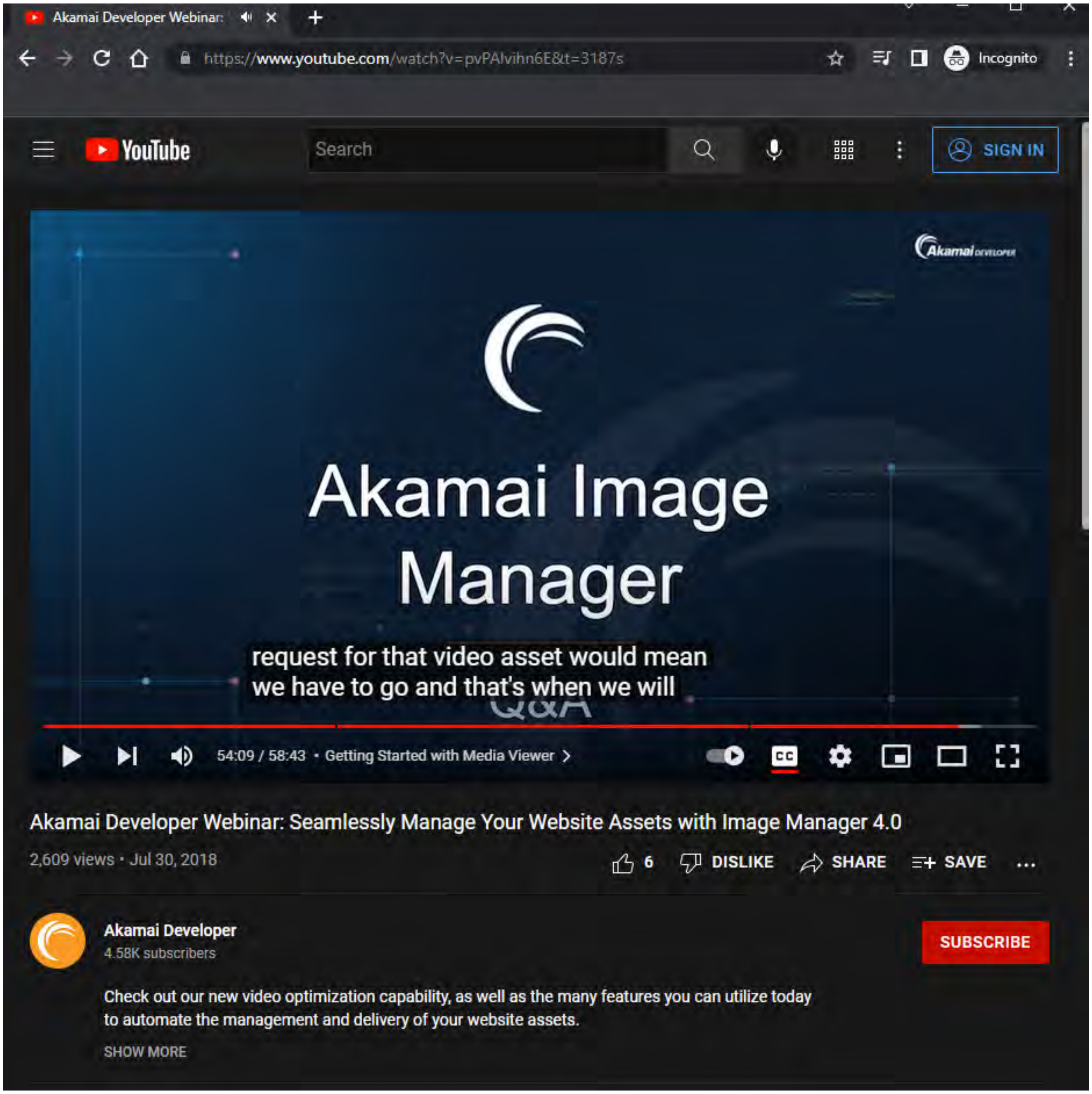
Akamai Developer
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

("Webinar – Image Manager 4.0")


Akamai Developer Webinar:
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
Q&A
54:09 / 58:43 • Getting Started with Media Viewer
6
DISLIKE
SHARE
SAVE
SUBSCRIBE
SHOW MORE

request for that video asset would mean we have to go and that's when we will

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

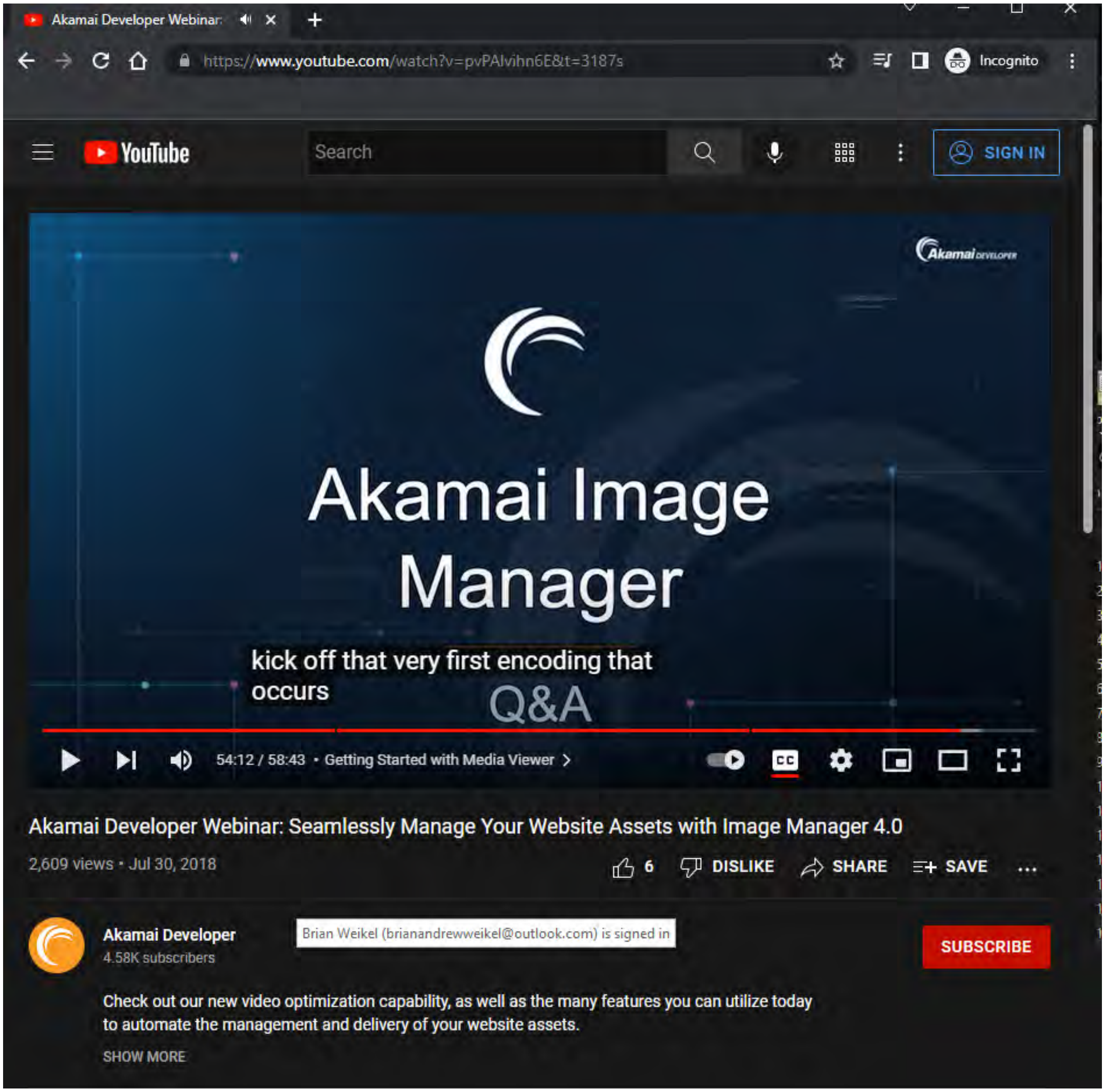
Akamai Developer
4.58K subscribers

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Akamai Developer Webinar Seamless Manage Your Website Assets with Image Manager 4.0

("Webinar – Image Manager 4.0")


Akamai Developer Webinar:
https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s
Incognito
YouTube
Search
SIGN IN
Akamai DEVELOPER
Akamai Image Manager
Q&A
54:12 / 58:43 • Getting Started with Media Viewer
6
DISLIKE
SHARE
SAVE
SUBSCRIBE
SHOW MORE

kick off that very first encoding that occurs

Akamai Developer Webinar: Seamlessly Manage Your Website Assets with Image Manager 4.0

2,609 views • Jul 30, 2018

Akamai Developer
4.58K subscribers

Brian Weikel (brianandrewweikel@outlook.com) is signed in

Check out our new video optimization capability, as well as the many features you can utilize today to automate the management and delivery of your website assets.

Source: https://www.youtube.com/watch?v=pvPAlvihn6E&t=3187s

Exhibit K

Adaptive Media Delivery

High-quality and secure streaming media delivery from Akamai's globally distributed content delivery network (CDN)

Akamai

Global online audiences demand instant, uninterrupted access to video on any device, anywhere. With numerous options for video consumption at their disposal, they have no qualms about turning to other providers to get the high-quality experience they demand. Meanwhile, the challenges of online content delivery are growing relentlessly, making it increasingly difficult to deliver consistent high-quality experiences to growing audience sizes amidst frequent spikes in demand.

Working with a partner that can take concerns about reliability, scalability, and security out of the equation, especially when there are sudden spikes in global viewership, is a necessity. Any misstep can have a long-lasting effect on audience perceptions and brand reputation.

Akamai's Adaptive Media Delivery

Akamai's Adaptive Media Delivery is optimized for adaptive bitrate (ABR) streaming to provide a high-quality viewing experience across the broad variety of network types – fixed or mobile – at varying connection speeds. Built on the Akamai Intelligent Edge Platform™, Adaptive Media Delivery provides superior scalability, reliability, availability, and reach.

The solution securely delivers prepared, pre-segmented HTTP-based live and on-demand streaming media, including support for the following video and music formats:

- HTTP live streaming (HLS)
- HTTP dynamic streaming (HDS)
- Microsoft smooth streaming (MSS)
- Dynamic adaptive streaming over HTTP (MPEG-DASH)
- Common media application format (CMAF)

While Adaptive Media Delivery is both workflow- and origin storage-agnostic, Akamai's complementary Media Services Live and NetStorage or Cloud Wrapper can be combined with Adaptive Media Delivery for an efficient, high-quality streaming media solution.

BENEFITS TO YOUR BUSINESS

HD **Maintain superior video quality and reliability** via Akamai's unique, globally distributed network architecture, which is backed by a 100% uptime SLA

Scale rapidly for large audiences – planned or unexpected – by leveraging the distributed network's ability to dynamically and intelligently distribute load

Optimize online video playback quality and performance through Akamai application-aware software intelligence, uniquely tuned to optimize ABR streaming

Reach audiences on any device with support for a wide range of standard video streaming protocols

Get actionable insights into streaming performance and quality of experience

Protect premium content and secure revenue streams

> "Our experience with Akamai is that they are a smart, passionate team that's about delivering results – the same parallel goals that we have at NBCSports.com and NBCSports Digital."
>
> – **Rick Cordella,**
> SVP Digital Media, NBC Sports Group

Adaptive Media Delivery

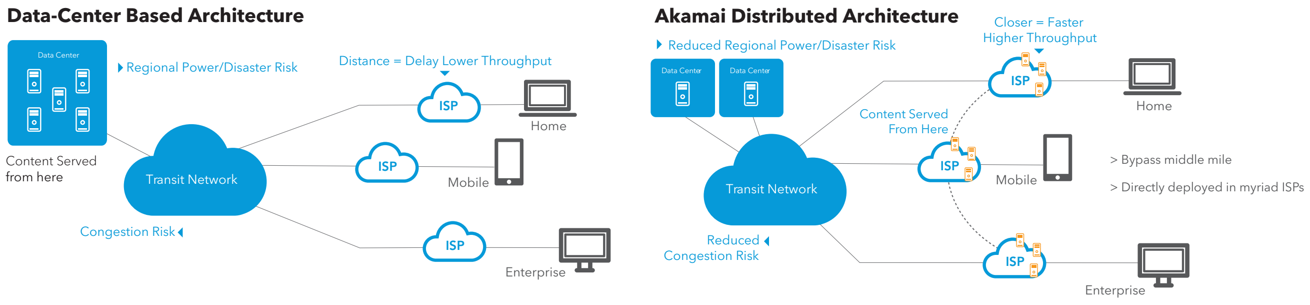
High-quality and secure streaming media delivery from Akamai's globally distributed CDN

Key Product Differentiators

Reliability, Capacity, and Scale: Deliver high-quality streams globally with a trusted partner that has set benchmarks for performance and reliability at scale. When a streaming event or over-the-top (OTT) service must succeed, the world's most successful brands turn to Akamai's Adaptive Media Delivery.

Quality Through Technology, Innovation, and Experience: Since 1998, Akamai's CDN market success has supported continuous investment in network capacity and technology innovations, to stay ahead of online audience demand.

Proximity – Closeness Counts: Even the highest-quality networks are subject to the Internet's realities of latency, congestion, and packet loss, which can significantly limit online video quality. Akamai has the most pervasive CDN, which minimizes the impact of network congestion for consistent viewing experiences. With approximately 253,000 servers in 137 countries and nearly 1,600 networks around the world, Akamai remains closer to the audience, for the lowest latency and highest quality.


Data-Center Based Architecture

Akamai Distributed Architecture

Akamai's distributed architecture and ISP partnerships place the servers – and your content – closer to the viewer for better reliability, availability, and performance. This approach offers the high-quality experience that online audiences increasingly expect.

Adaptive Media Delivery

High-quality and secure streaming media delivery from Akamai's globally distributed CDN

Standard and Optional Features

RELIABILITY AND SCALE

Akamai Intelligent Edge Platform
Delivers on average 50 Tbps of Internet traffic daily, with a record of more than 80 Tbps

100% Uptime SLA
Will serve content 100% of the time

Origin Failover
Ensure uptime by switching from one origin to another if a failure occurs including configurable policies, recovery methods and failure reporting for a variety of scenarios.

PERFORMANCE

Advanced Cache Control
Intelligently caches content close to end users while offloading traffic on the origin infrastructure

Tiered Distribution
Additional caching layer further optimizes origin offload and improves performance for end users

ABR Streaming
Adjusts content dynamically for changing network conditions to ensure a high-quality experience

Segment Prefetch
Optimizes video playback quality by utilizing origin assistance to pre-position anticipated video segment at the edge during playback

Quick Retry
Enhances delivery performance through detection of Internet bottlenecks, and attempts alternate paths to avoid slower connections

HTTP/3 QUIC Support
A new UDP-based transport protocol that can improve video performance and provide a consistent quality of experience

HTTP/2
New protocol provides many improvements over previous versions in terms of efficiency, performance, and security

IPv6
Support for the latest communication protocol, including dual stack IPv4/IPv6

Purge & Fast Metadata Activation
Activate property configuration changes and update content rapidly

SECURITY

Standard TLS
HTTPS delivery over custom certs using SNI

TLS: Shared Cert
HTTPS delivery using a SAN certificate from Akamai and a shared domain

Access Control
Manages security controls that govern end-user access to content

Token Authentication
Secures premium content from unauthorized access by preventing link sharing

Cloud Authentication
Delivers content from various cloud origins using simple authentication configurations

Adaptive Media Delivery

High-quality and secure streaming media delivery from Akamai's globally distributed CDN

CUSTOMER UI AND SELF-SERVICEABILITY

Property Manager
A guided, self-service configuration tool within the Akamai Control Center that enables customers to control and manage their products and services

Log Delivery
Access to logs generated from traffic on the Akamai Intelligent Edge Platform

Media Reports
Insights into delivery performance and usage

Akamai Control Center
Provides customers a portal to manage all Akamai products and services

Event Center
Provides insight into live events by streamlining planning, management, and reporting

APIS AND OTHER FEATURES

Open APIs
Operate and integrate with the Akamai Intelligent Edge Platform via https://developer.akamai.com

Beta Channel
An opt-in program allowing customers to quickly and easily adopt and evaluate new Akamai beta capabilities

Third-Party Origin Support
Leverage any origin storage of your choice

OPTIONS

Security

Enhanced Proxy Detection
Prevents content piracy through use of an integrated VPN and DNS proxy detection service

Enhanced TLS
Offers added security measures and provides HTTPS delivery using a custom certificate

Edge IP Binding
Enables whitelisting of a limited number of IP addresses for:
- Monitoring traffic usage associated with zero-rated billing agreements
- Granting access behind a network firewall

Content Targeting
Restricts access to content based on geographic location

Media Encryption
Protects content against unauthorized viewing using scalable session-level encryption

Protocol Downgrade
Deliver content using the secure HTTPS protocol even when it is not supported by an origin infrastructure

Watermarking Support
Scalable session-based forensic watermarking support from the Akamai Edge, pre-integrated with several leading third-party providers, which enables content providers to protect premium video assets by tracing piracy back to the source

Access Revocation
An API that can be used to shut down access to a video stream in near real time to protect premium content against unauthorized viewing and piracy

Other Features

Manifest Personalization
Personalize video quality for each viewer based on device type, network conditions, or geography to ensure optimal playback

Dynamic Ad Insertion
Customers are able to serve highly relevant, targeted ads to individual viewers

China CDN
Delivery of content into China while ensuring full compliance with all Chinese government licensing requirements

Cloud Interconnects
Provides optimal connections with public cloud providers to ensure high-performance data transfer, and in some instances reduces cloud egress costs

Adaptive Media Delivery

High-quality and secure streaming media delivery from Akamai's globally distributed CDN

Akamai Ecosystem

Akamai makes the Internet fast, reliable, and secure. Our comprehensive solutions are built on the globally distributed Akamai Intelligent Edge Platform; managed through the unified, customizable Akamai Control Center for visibility and control; and supported by professional services experts who get you up and running easily and inspire innovation as your strategies evolve.

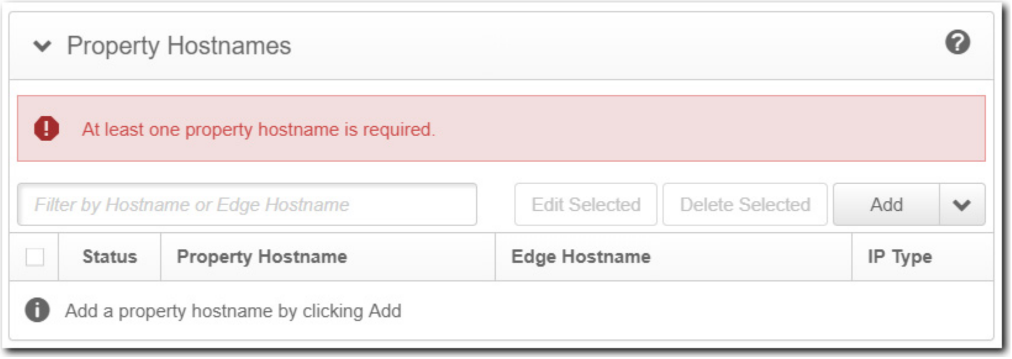
Exhibit L

(Reserved)

Exhibit M

Define property hostnames

You use this content panel to create your property hostname associations. These associations play a key role in getting your streaming media out to Akamai edge servers, for access to requests from your end users.


Property Hostnames

At least one property hostname is required.

□	Status	Property Hostname	Edge Hostname	IP Type

Add a property hostname by clicking Add

Understand the property hostname association

Essentially, you use the hostname for your streaming media to create an "edge hostname" that's used to get your Adaptive Media Delivery property to determine how to deliver your streaming media.

- **The hostname**. During the property hostname process, you include this to verify that a request belongs to you as our customer, as well as determine which Property Manager configuration ("property") to use for delivery settings. This is typically the full hostname or domain name that end users target to access your media, without the `https://` . For example, `www.sportsvideos.baseball.hof-induction.mp4` , `sailing.clipperships.m4v` , and `my.travel.videos.com` are examples of hostnames you could use. This is also referred to as the "vanity hostname."
- **The edge hostname**. Property Manager automatically creates an edge hostname, using the hostname you provide. An edge hostname uses the "Akamai Intelligent Platform" to map requests for your media to Akamai edge servers. For example, if you provided the hostname `my.travel.videos.com` for your site, Property Manager creates the edge hostname, `my.travel.videos.com.akamaized.net` .

Later, when you're ready to go live, you change the DNS record for your hostname to "CNAME" to the edge hostname to acknowledge this association. End-user requests to your hostname are redirected to the edge hostname. These resolve to Akamai edge servers that read from your Adaptive Media Delivery property to determine how to deliver your media. The actual delivery is based on rules and behaviors you establish in your Adaptive Media Delivery property.

Edge hostnames work behind the scenes and are not visible in the URLs of content served to end users.

Your content is cached on edge servers

You can set various caching rules and behaviors in your Adaptive Media Delivery property, and some are automatically applied as best practices. When a request for your media content goes through your edge hostname and it's processed by your property, that content is held in cache on the Akamai edge network. This reduces the need to access your origin and can also speed up delivery to other customers, that also make requests to the same edge hostname.

Set up secure authentication

If you haven't already, you need to set up a certificate if you want to deliver your site or app via HTTPS.

Set up a secure property hostname

You use Property Manager for this, and the process is predominantly the same regardless of product. To maintain consistent instructions, it's covered in the Property Manager documentation.

> **Non secure property hostnames**
> The standard practice is to support secure access and this is what Akamai recommends. But, Property Manager also offers multiple ways to create non-secure (HTTP) property hostnames, too.

Updated 2 months ago

← Create a new AMD property

Define property variables (optional) →

Did this page help you? Yes No

Exhibit N

Akamai edge
2012 CUSTOMER CONFERENCE

Transcoding in the Cloud

David Bornstein - Director, Product Line Management, Sola Media
Christopher DeGrace - Product Manager, Sola Media
Kurt Michel - Director, Product Marketing, Sola Media

Akamai
FASTER FORWARD

Why Transcode?

- Deliver high quality experiences using adaptive bitrate (ABR) streaming

- Reach as many devices as possible, given varying network and device capabilities

- Save network bandwidth and transfer time
- Rightsizing large "Master" file sizes/bitrates for online delivery

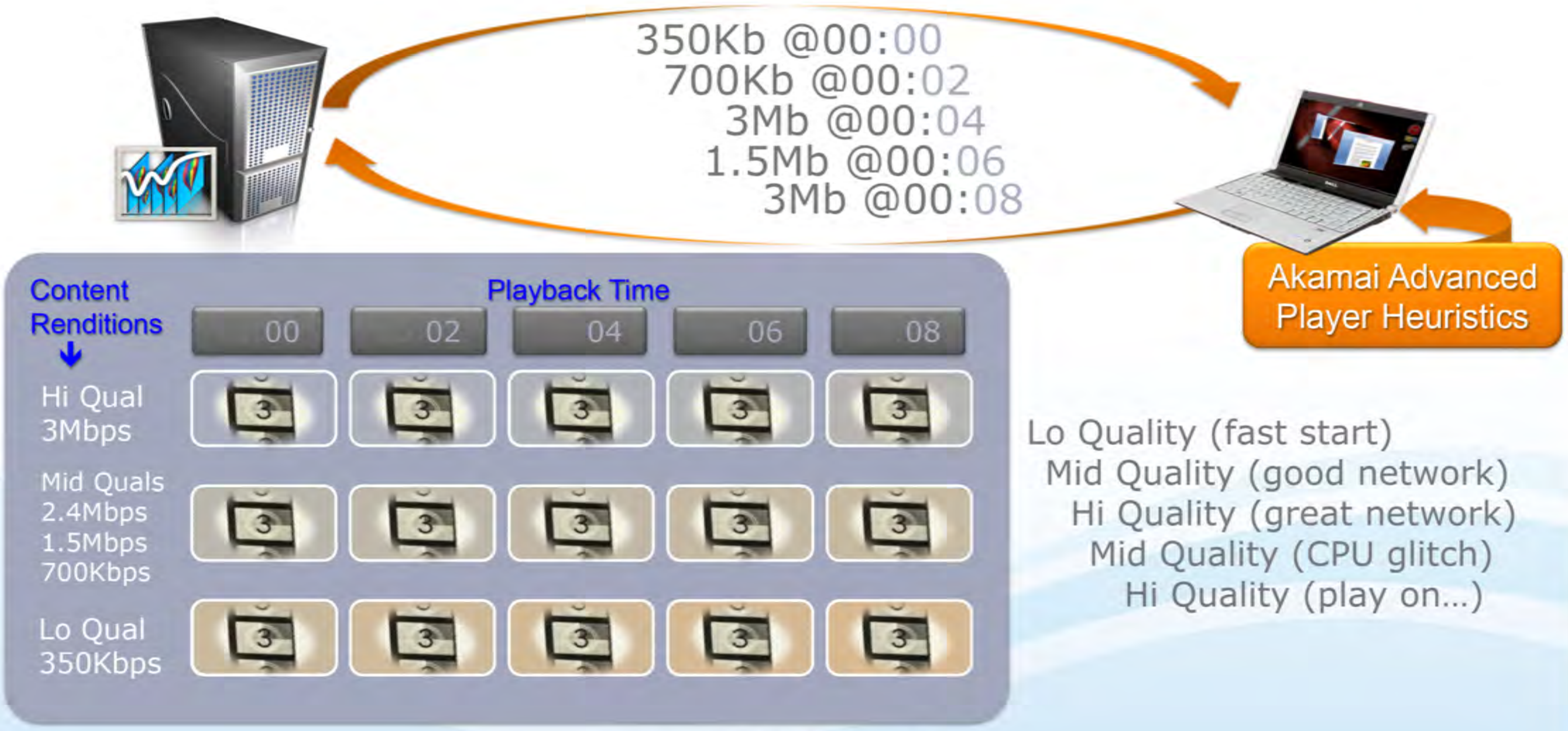
- Future-Proof
- Content created today can be transcoded to better codec technology (same quality, smaller size file) in the future.

With Transcoding

~~Or~~ AND this

EASY Choice!

Transcoding: Key to ABR Streaming Success



Lo Quality (fast start)
Mid Quality (good network)
Hi Quality (great network)
Mid Quality (CPU glitch)
Hi Quality (play on...)

Transcoding Market

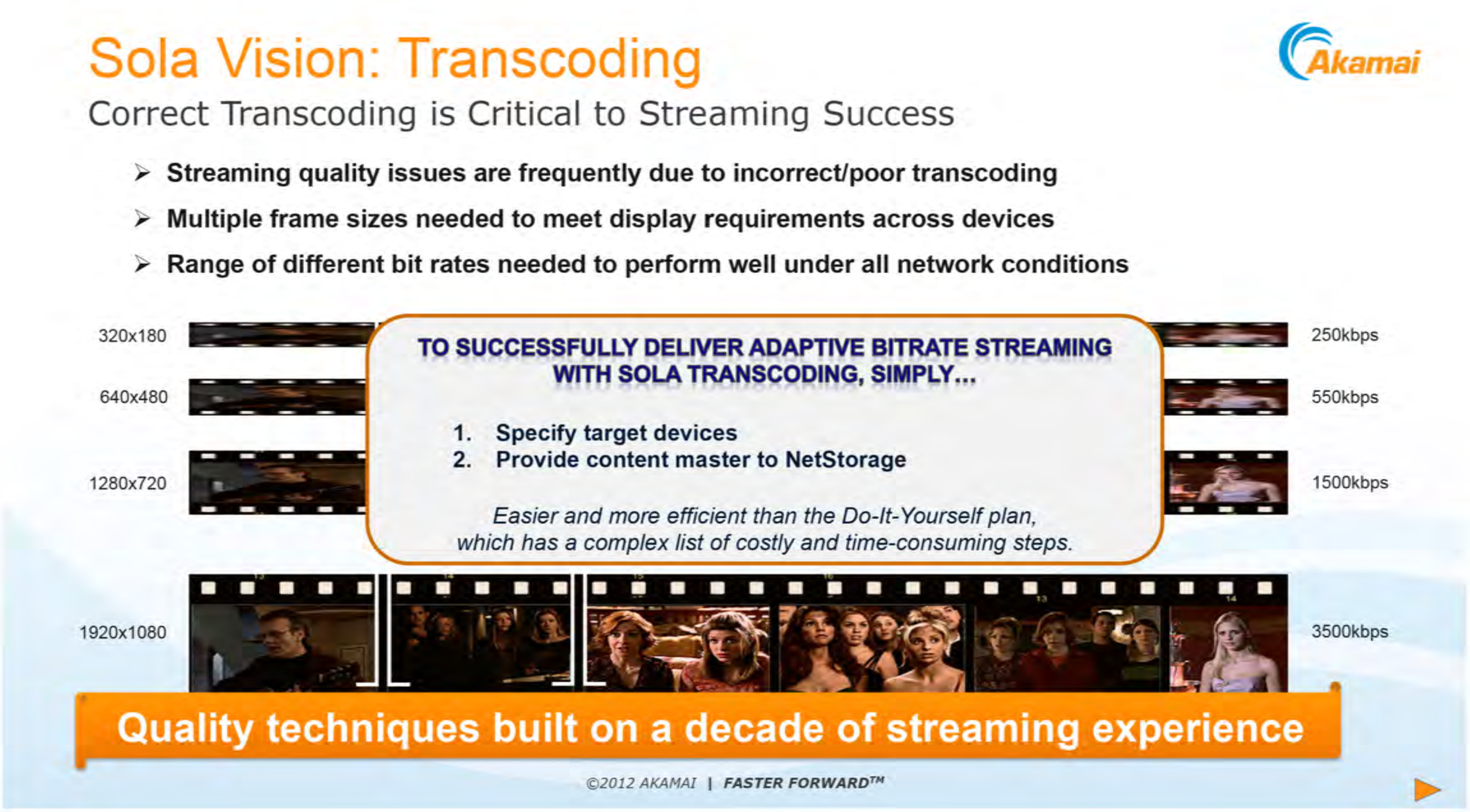
Akamai

FROST & SULLIVAN

Video transcoders market earned $230M in 2010 and is expected to exceed $750M in 2017

In-Stat
An NPD Group Company

Video transcoding to hit $460M business by 2015



Sola Vision: Transcoding

Correct Transcoding is Critical to Streaming Success

- Streaming quality issues are frequently due to incorrect/poor transcoding
- Multiple frame sizes needed to meet display requirements across devices
- Range of different bit rates needed to perform well under all network conditions

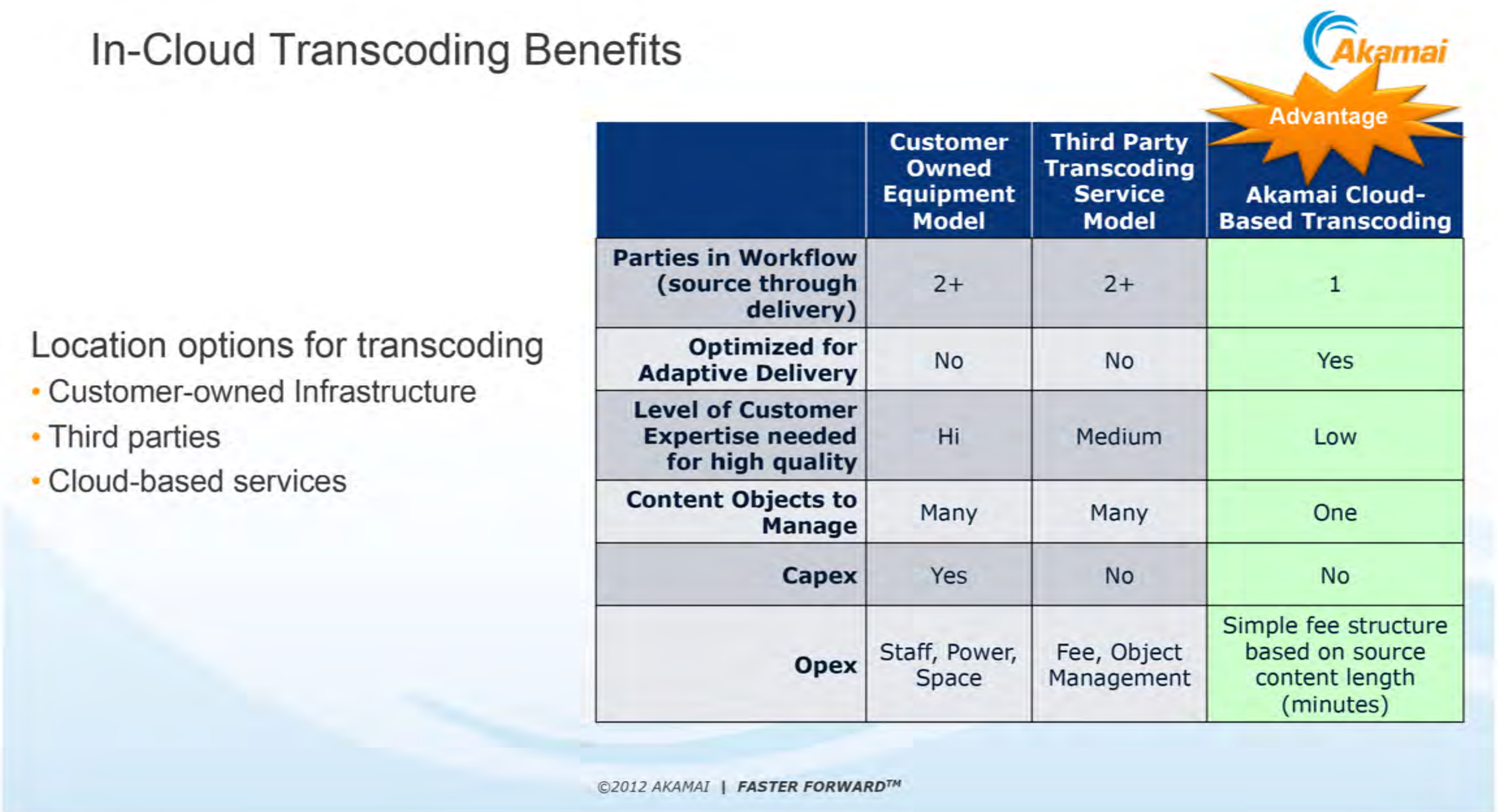
320x180	250kbps
640x480	550kbps
1280x720	1500kbps
1920x1080	3500kbps

TO SUCCESSFULLY DELIVER ADAPTIVE BITRATE STREAMING WITH SOLA TRANSCODING, SIMPLY…

1. Specify target devices
2. Provide content master to NetStorage

Easier and more efficient than the Do-It-Yourself plan, which has a complex list of costly and time-consuming steps.

Quality techniques built on a decade of streaming experience



In-Cloud Transcoding Benefits

Location options for transcoding
- Customer-owned Infrastructure
- Third parties
- Cloud-based services

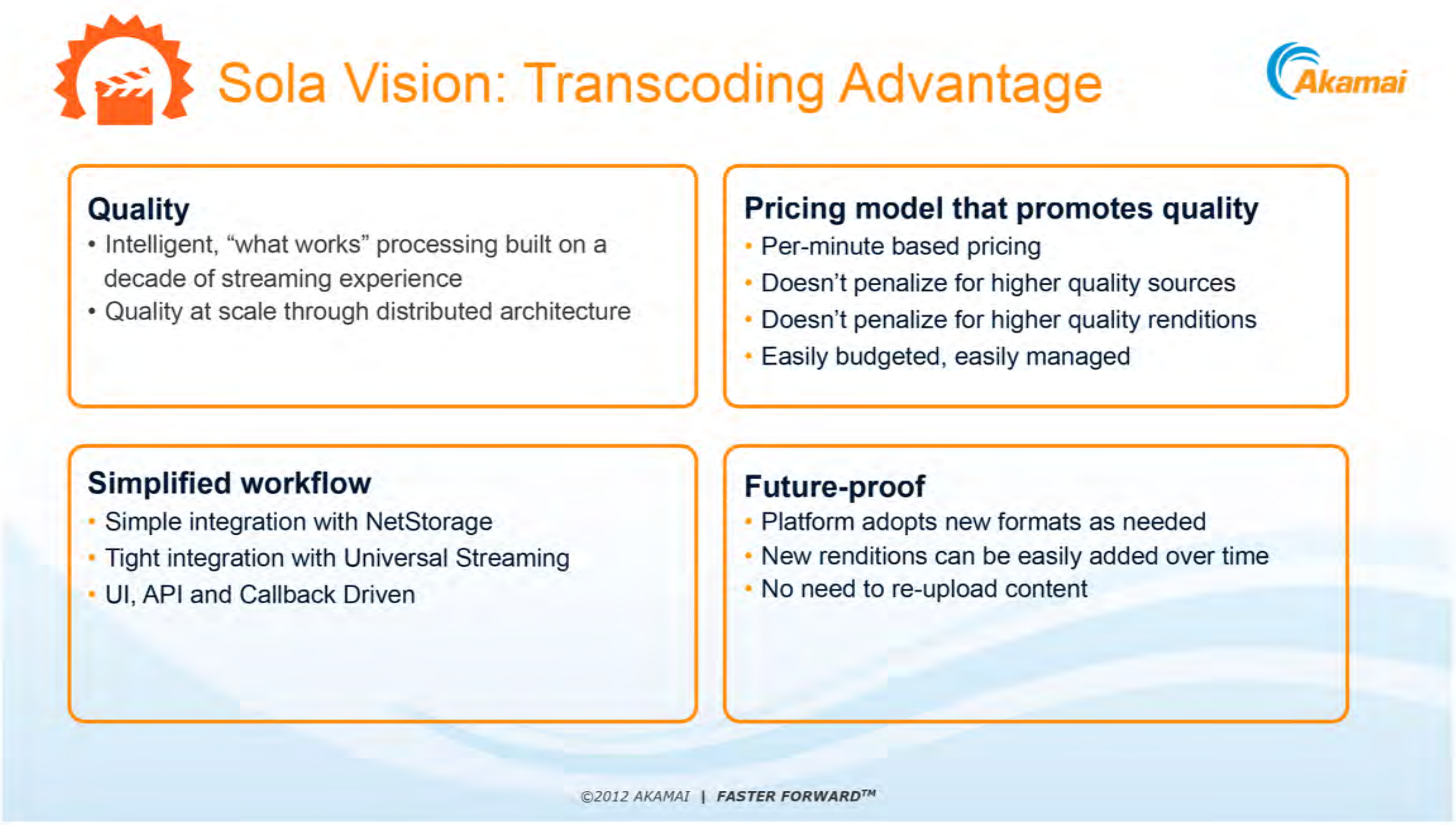
Advantage

	Customer Owned Equipment Model	Third Party Transcoding Service Model	Akamai Cloud-Based Transcoding
Parties in Workflow (source through delivery)	2+	2+	1
Optimized for Adaptive Delivery	No	No	Yes
Level of Customer Expertise needed for high quality	Hi	Medium	Low
Content Objects to Manage	Many	Many	One
Capex	Yes	No	No
Opex	Staff, Power, Space	Fee, Object Management	Simple fee structure based on source content length (minutes)

Akamai

Transcoding in the Cloud

Sola Vision



Sola Vision: Transcoding Advantage

Akamai

Quality

- Intelligent, “what works” processing built on a decade of streaming experience
- Quality at scale through distributed architecture

Pricing model that promotes quality

- Per-minute based pricing
- Doesn’t penalize for higher quality sources
- Doesn’t penalize for higher quality renditions
- Easily budgeted, easily managed

Simplified workflow

- Simple integration with NetStorage
- Tight integration with Universal Streaming
- UI, API and Callback Driven

Future-proof

- Platform adopts new formats as needed
- New renditions can be easily added over time
- No need to re-upload content

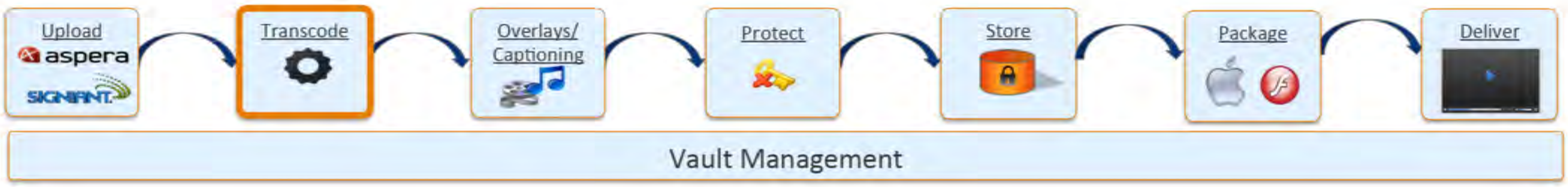
Akamai edge
2012 CUSTOMER CONFERENCE

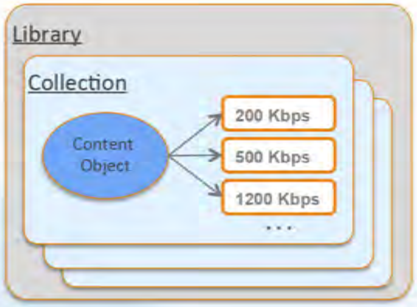
Played Transcoding Lego Video

Akamai
FASTER FORWARD

VoD Transcoding Overview – Preparation Workflow

Akamai





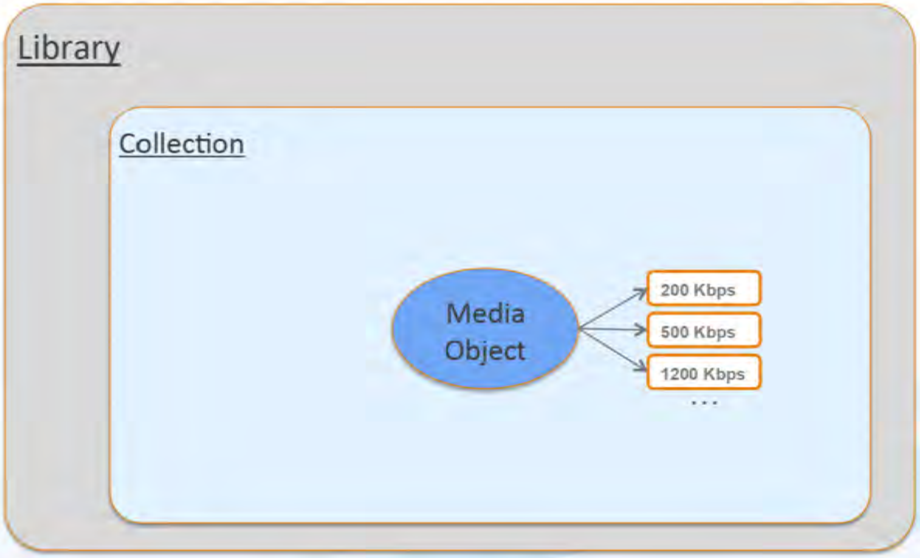
Transcoding for delivery to Devices

- Facilitation of the Media Preparation Workflow
- Transform, Protect, Store and Deliver
- Simplified Management of Assets

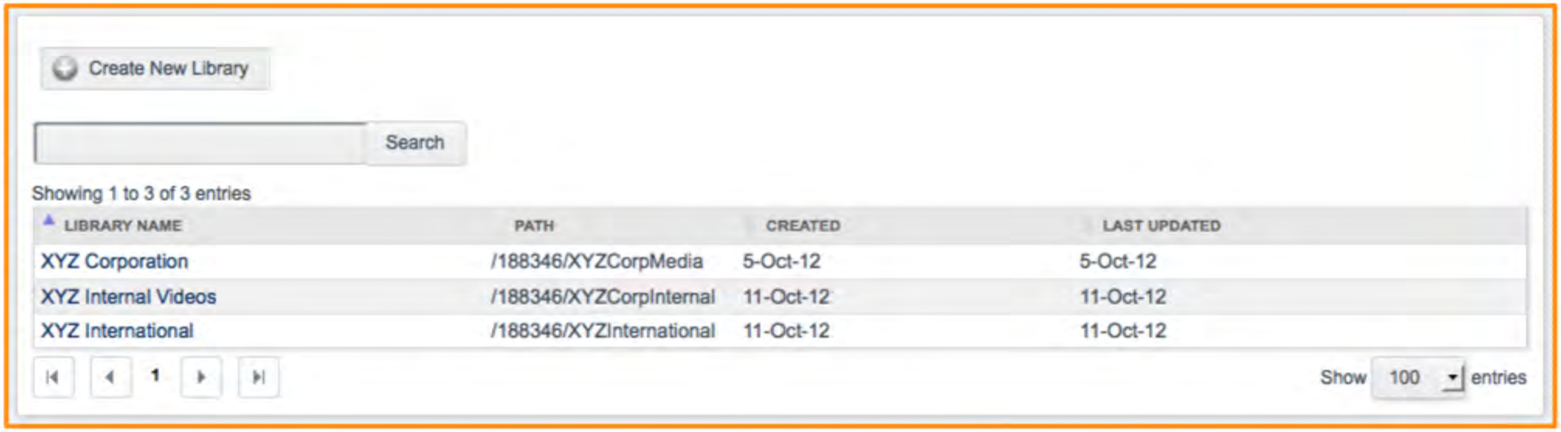
The Transcoding Building Blocks

Akamai

- Libraries
- Collections
- Media Objects



Media Libraries

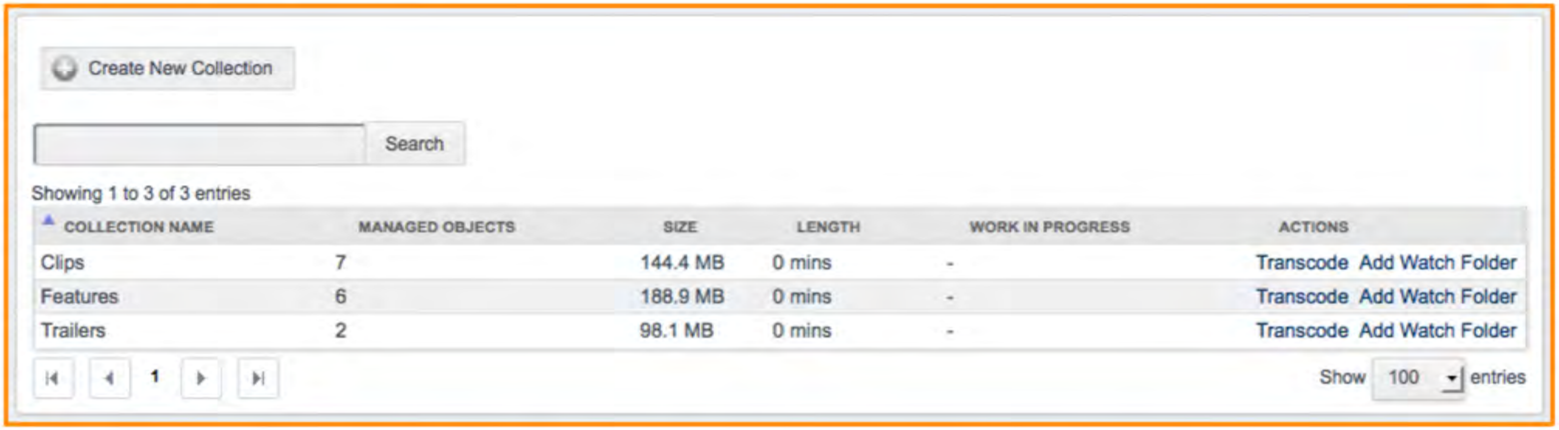


Showing 1 to 3 of 3 entries

LIBRARY NAME	PATH	CREATED	LAST UPDATED
XYZ Corporation	/188346/XYZCorpMedia	5-Oct-12	5-Oct-12
XYZ Internal Videos	/188346/XYZCorpInternal	11-Oct-12	11-Oct-12
XYZ International	/188346/XYZInternational	11-Oct-12	11-Oct-12

- Manage multiple Libraries for different projects, groups, divisions, etc.
- Libraries define the prioritization queues, storage, delivery, analytics and billing

Media Collections



Showing 1 to 3 of 3 entries

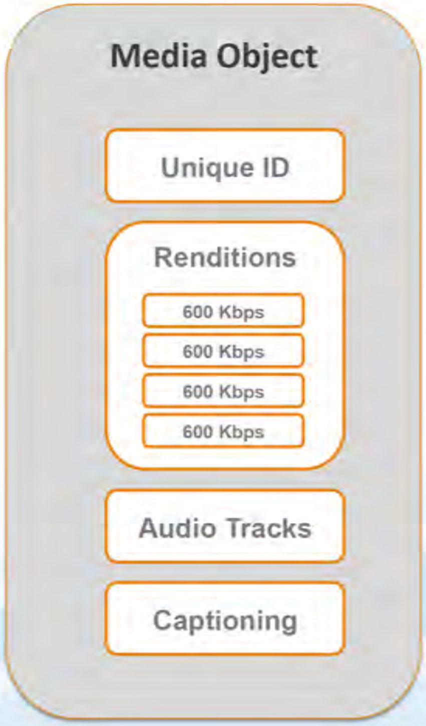
COLLECTION NAME	MANAGED OBJECTS	SIZE	LENGTH	WORK IN PROGRESS	ACTIONS
Clips	7	144.4 MB	0 mins	-	Transcode Add Watch Folder
Features	6	188.9 MB	0 mins	-	Transcode Add Watch Folder
Trailers	2	98.1 MB	0 mins	-	Transcode Add Watch Folder

- Manage different types of content within your group independently
- Transcoding Profiles are assigned to the collections
- Update, Adapt, Re-transcode groups of videos within a collection

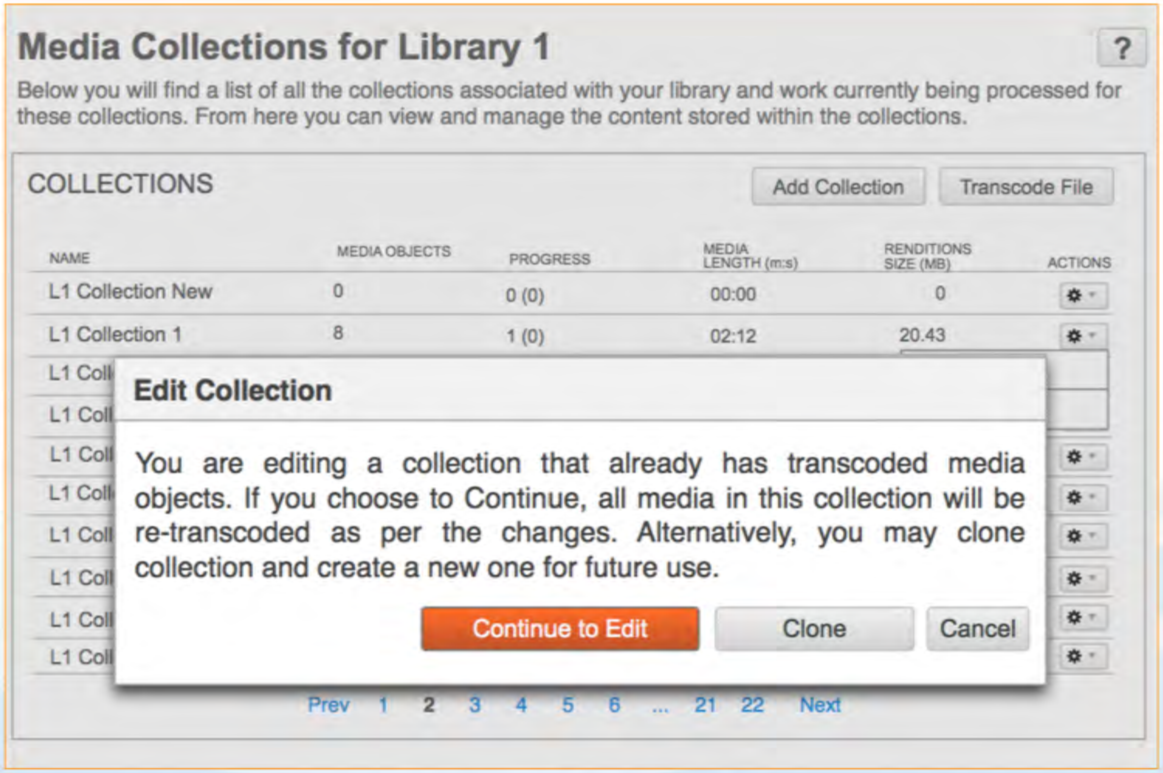
Media Objects

- Videos are managed as Media Objects
- Media Objects contain the video-centric metadata about the object
- Identified by a Unique Identifier

Note: Akamai is NOT building a CMS System



Collection Management



Media Collections for Library 1

Below you will find a list of all the collections associated with your library and work currently being processed for these collections. From here you can view and manage the content stored within the collections.

COLLECTIONS

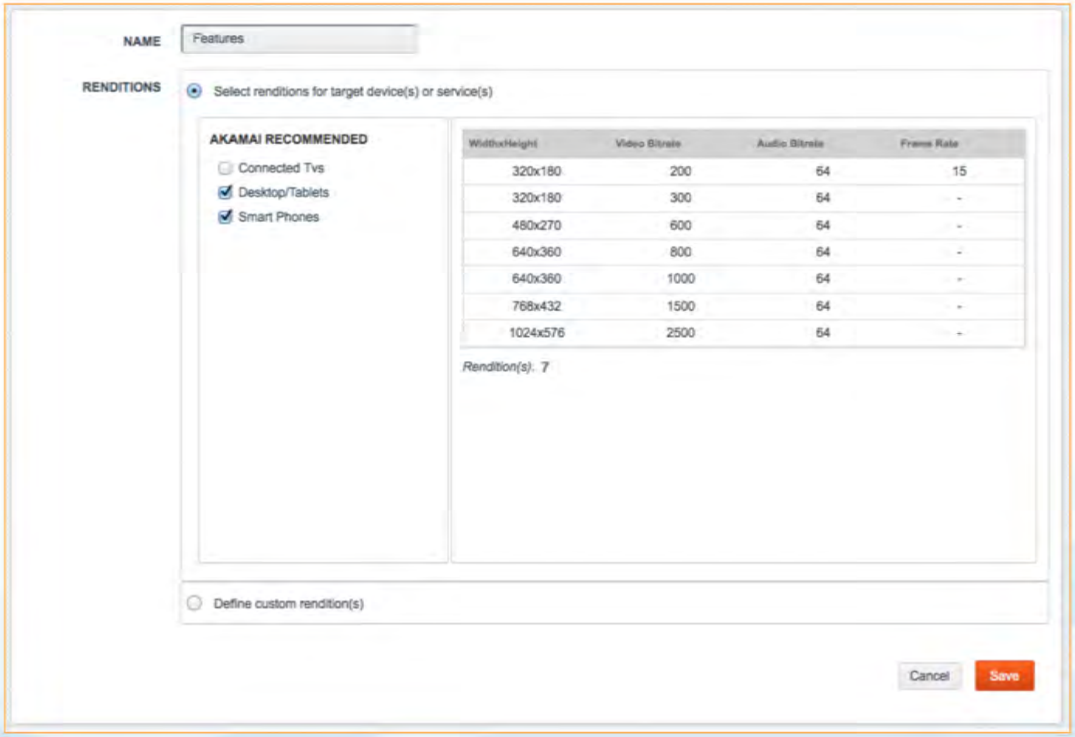
NAME	MEDIA OBJECTS	PROGRESS	MEDIA LENGTH (m:s)	RENDITIONS SIZE (MB)	ACTIONS
L1 Collection New	0	0 (0)	00:00	0	✲ ▾
L1 Collection 1	8	1 (0)	02:12	20.43	✲ ▾
L1 Coll					
L1 Coll					
L1 Coll					✲ ▾
L1 Coll					✲ ▾
L1 Coll					✲ ▾
L1 Coll					✲ ▾
L1 Coll					✲ ▾
L1 Coll					✲ ▾

Edit Collection

You are editing a collection that already has transcoded media objects. If you choose to Continue, all media in this collection will be re-transcoded as per the changes. Alternatively, you may clone collection and create a new one for future use.

Profile Configuration

Simple



NAME: Features

RENDITIONS

● Select renditions for target device(s) or service(s)

AKAMAI RECOMMENDED

□ Connected Tvs
☑ Desktop/Tablets
☑ Smart Phones

WidthxHeight	Video Bitrate	Audio Bitrate	Frame Rate
320x180	200	64	15
320x180	300	64	-
480x270	600	64	-
640x360	800	64	-
640x360	1000	64	-
768x432	1500	64	-
1024x576	2500	64	-

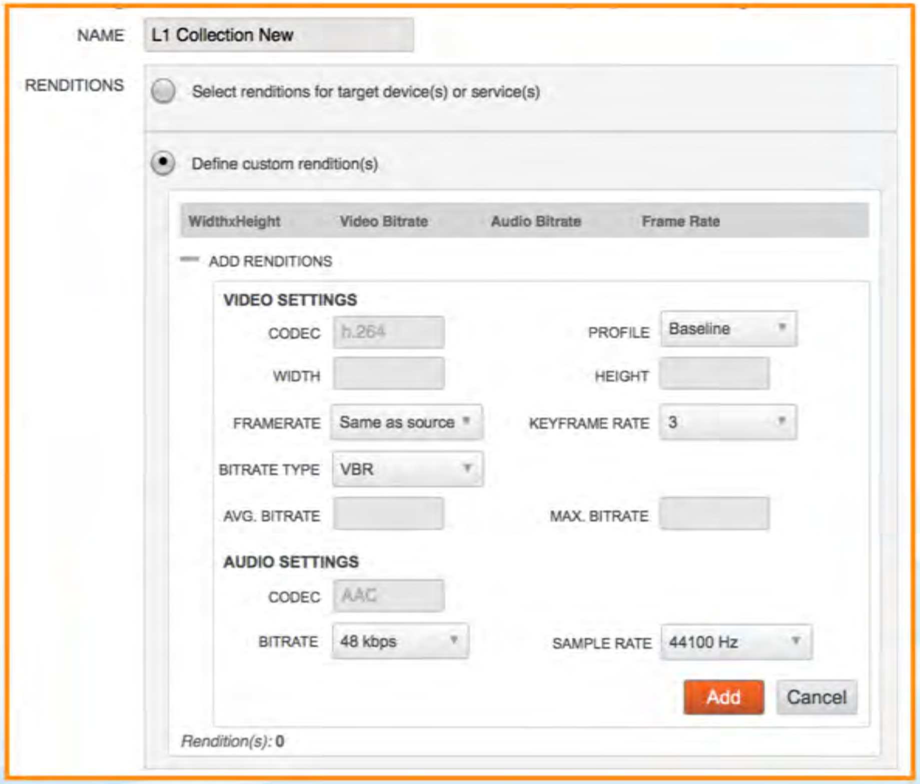
Rendition(s): 7

○ Define custom rendition(s)

Profile Configuration

Flexible

Advanced Options also available



NAME: L1 Collection New

RENDITIONS

○ Select renditions for target device(s) or service(s)

● Define custom rendition(s)

WidthxHeight	Video Bitrate	Audio Bitrate	Frame Rate

— ADD RENDITIONS

VIDEO SETTINGS

CODEC	h.264	PROFILE	Baseline
WIDTH		HEIGHT	
FRAMERATE	Same as source	KEYFRAME RATE	3
BITRATE TYPE	VBR		
AVG. BITRATE		MAX. BITRATE	

AUDIO SETTINGS

CODEC	AAC		
BITRATE	48 kbps	SAMPLE RATE	44100 Hz

Rendition(s): 0

Exhibit O

Akamai

Building an Ecosystem: Transcoding for 24x7 Live Linear

A COMPLEX ECOSYSTEM

With the proliferation of viewing platforms, file formats and streaming technologies competing in today's online media ecosystem, video transcoding is becoming increasingly complicated and cumbersome. Efficiently managing cloud transcoding has become paramount in an increasingly complex and fragmented device ecosystem. As such, most cloud transcoding discussions have revolved around rapidly adding new capabilities to support the surge in formats and codecs, while also simplifying workflows.

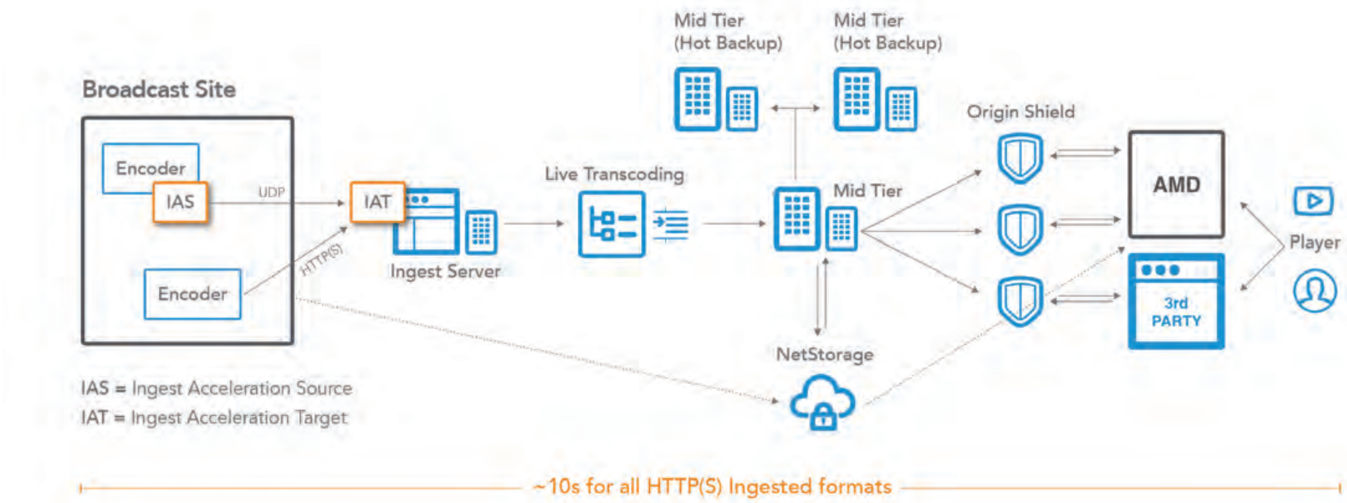
However, as the value of content services increases and businesses look towards launching their own 24x7 live linear offerings, the operational impact of managing a transcoding system also becomes highly critical. Transcoding solutions now need to provide support in a 24x7x365 environment, while delivering a TV-like experience to online audiences.

TRANSCODING IN A 24X7 LIVE LINEAR ENVIRONMENT

A live transcoding system undoubtedly needs to support a variety of formats and bitrates, and be able to optimize picture quality to match user expectations. For cloud transcoding services of the future to truly serve the growing 24x7 live linear market, the conversation needs to shift to how transcoding fits into a larger ecosystem. Content providers must consider other components of the ecosystem that help provide a quality viewing experience for their audiences, including:

1. Reliable ingest of streams
2. Low-latency support
3. 24x7 availability and redundancy
4. Visibility
5. Formats and quality

A Transcoding System Designed to Enable 24x7 Live Linear Streaming



Akamai

Knowing the importance of a high-performance ecosystem, Akamai developed live transcoding functionality after ensuring its full integration into our liveOrigin™ solution. liveOrigin™ is a collection of capabilities that encompasses all the components listed above and was designed to overcome the challenges of delivering broadcast-quality 24x7 live linear streams.

A high-performance transcoding solution, complemented with the capabilities listed above, is what would enable content providers to truly deliver differentiated value to their online audiences.

1) RELIABLE INGEST OF STREAMS

Quality starts in the first mile. The best live transcoding system won't provide much value unless content is reliably ingested from the encoder. Akamai's ingest acceleration technology uses proprietary UDP transport protocol to improve performance and throughput of content when transferring streams into the cloud.

The Ingest Acceleration Source (IAS) software (downloaded through Akamai's customer portal) takes the stream from the customer's encoder, and forwards it over our proprietary UDP transport protocol. It is received by the Ingest Acceleration Target (IAT), where the stream is decoded into it's original format and handed to the entry point software for further processing.

This process enables fast, consistent, and reliable ingest of content. UDP technology is also used to improve the quality of the live feed even in cases of packet loss to end-user devices through Akamai's Delivery platform.

2) LOW-LATENCY SUPPORT

Until now, the online experience could be delayed anywhere from to 30 seconds to 2 minutes behind broadcast. To deliver a quality experience to their online viewers, content providers need to ensure that their transcoders aren't adding additional latency to their live linear streams.

Akamai supports 10-second end-to-end, hand-wave latency for live linear and live streams through combining several key capabilities that span from ingest to Edge servers. Several of these capabilities are as follows:

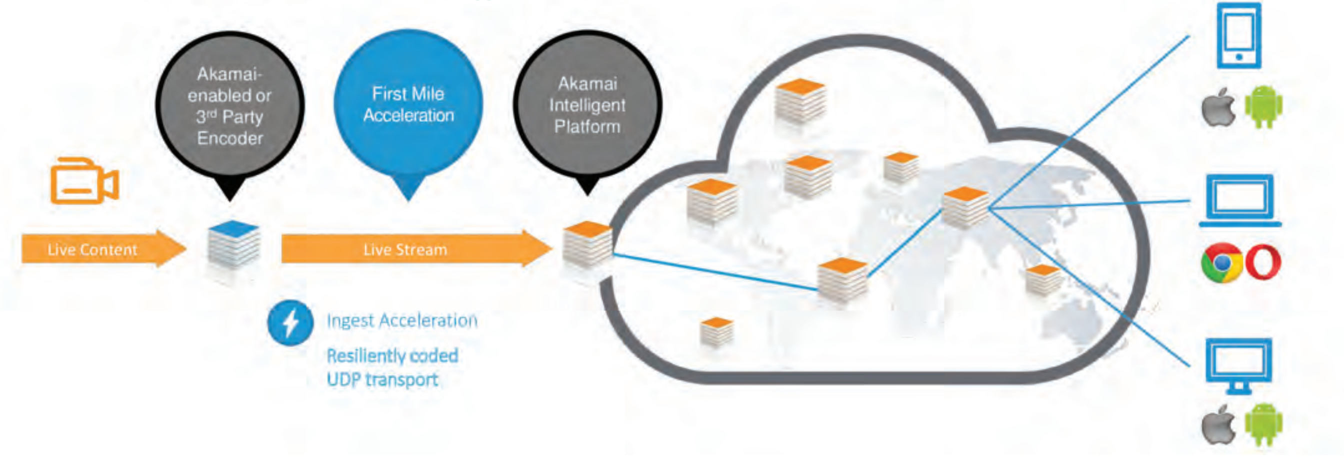
- **Live transcoding in real time**: A live transcoding system needs to elastically scale to handle peak loads of video in real-time. Any latency added during transcoding is ultimately passed on to the end user, resulting in a less than optimal viewing experience.

- **Small segment sizes:** Akamai's architecture was built to reliably handle small segment sizes (down to 2-seconds) for HTTP-based streaming to enable players to quickly switch down on bandwidth drops and prevent player stalls and effectively reduce client-side buffers.

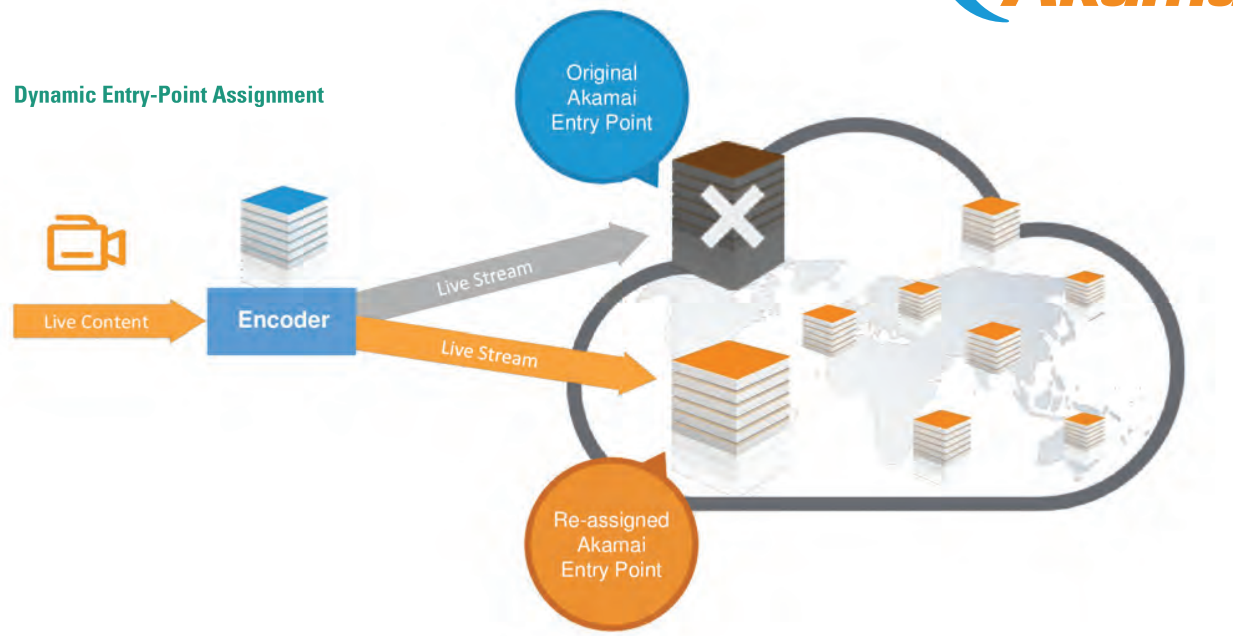
- **HTTP chunked encoded transfers:** Supporting chunked encoded transfers from ingest through to the Edge to initiate transfers as soon as the data is available helps minimizes latencies.

- **Prefetching from the Edge:** Edge servers will pre-fetch the next set of segments for a particular bitrate and cache it locally as the previous segment

Supporting 10-second, hand-wave latency


Akamai-enabled or 3rd Party Encoder
First Mile Acceleration
Akamai Intelligent Platform
Live Content
Live Stream
Ingest Acceleration
Resiliently coded UDP transport

Dynamic Entry-Point Assignment



is already received and being played. This allows segments to be readily available and reduces risk of additional latency.

3) 24X7 AVAILABILITY AND REDUNDANCY

Video failures have a tangible, negative impact on viewers and their future perceptions of brands - Only 8.2% of viewers returning to a site within 24 hours after a video failure. Content providers need to ensure their cloud-based live transcoding system is operating at near-perfect availability and redundancy.

There are several capabilities Akamai utilizes to achieve unparalleled 24x7 reliability and redundancy through enabling flexible allocation of resources and routing of video assets:

- **Segment replication:** Akamai creates multiple copies of each segment upon ingest, enabling the transcoding system to access the assets from multiple locations. Each segment can also be pulled from several different locations when the transcoding system prepares the video. In the event the transcoding systems fails to retrieve the asset from one location, it reroutes to fetch segments from another source.

- **Load balancing:** Once the segment enters the transcoding system, Akamai's load balancing system optimizes how segments are sent to transcoding resources.

- **Dynamic entry-point assignment:** Akamai Entry Points use dynamic assignment to assign encoders to an optimal Entry-Point. The same mapping technology also allows Akamai to reroute streams to a new Entry Point should network conditions change.

4) VISIBILITY

Understanding how a 24x7 live linear stream is performing empowers content providers to quickly identify and mitigate issues. Near real-time monitoring and insights into the transcode process, complemented by visibility into stream health and first-mile performance, is key to optimize processes and take corrective action as needed. Akamai provides near-real time visibility into critical first-mile performance metrics, such as:

- Latency
- Number of current streams
- Bitrates being ingested
- Fluctuations over the past hour
- Packet loss
- Errors

FORMATS AND QUALITY

Audience expectations are rising and quality issues pose the biggest threat to attracting and retaining subscribers and viewers of an OTT service. An independent study conducted by Columbia University on why Quality of Experience (QoE) is more important than Quality of Service (QoS), the importance of

Akamai

delivering a smooth viewing experience was stark. Abandonment rates increased by more than 4x with when bitrates shifted in comparison to when kept constant (**go2sm.com/columbiauniversity**).

At this point it should be clear that having the most sophisticated transcoding system on the market by itself can't deliver a high quality 24x7 live linear stream viewers in isolation. The reverse is also true. The capabilities discussed above are relatively meaningless unless they work in concert with a transcoding engine to help deliver the best experience possible to online audiences across a wide array of formats and devices.

To help overcome quality challenges, Akamai's Live Transcoding supports:

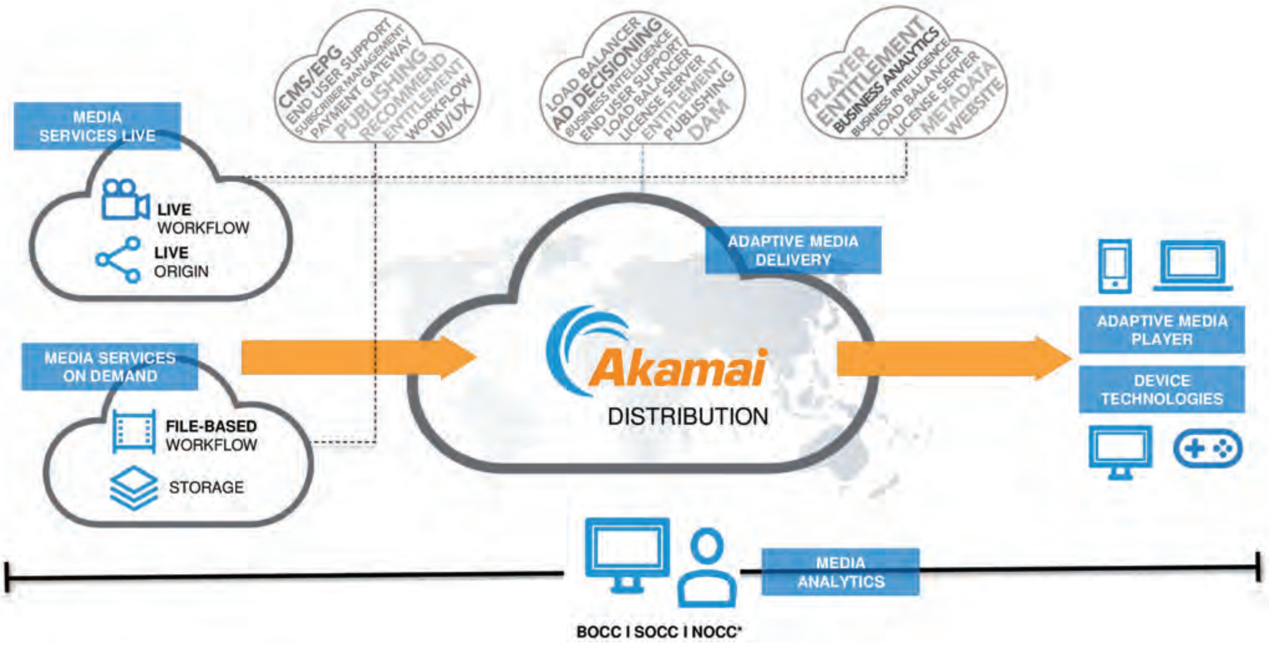
- Ingestion of bitrates up to 20Mbps from a single HLS stream for maximum output quality
- Default support for 1080p video formats to enable the best picture quality for end-viewers
- Support for leading adaptive bitrate formats
- Optimized transcoding profiles designed to reduce total bandwidth while preserving picture quality

Reach out to us at **www.Akamai.com** or contact your Akamai rep for more information on our 24x7 Live Linear solution with Media Services Live.

ABOUT AKAMAI TECHNOLOGIES

As the global leader in Content Delivery Network (CDN) services, Akamai makes the Internet fast, reliable and secure for its customers. The company's advanced web performance, mobile performance, cloud security and media delivery solutions are revolutionizing how businesses optimize consumer, enterprise and entertainment experiences for any device, anywhere. To learn how Akamai solutions and its team of Internet experts are helping businesses move faster forward, please visit **www.akamai.com** or **blogs.akamai.com**, and follow **@Akamai** on Twitter.

The Akamai Ecosystem



****BOCC** is our Broadcast Operations Control Center is managed solution that provides direct access to Akamai media experts who offer hands-on assistance for customers looking to offload (or supplement) proactive monitoring, alerting, live support and mitigation for their OTT video streams.*

***NOCC** is the Network Operation Command Center is a state-of-the-art facility where Akamai monitors the industry's best view of the real-time condition of the Internet. The NOCC is staffed 24 hours a day, 7 days a week by expert network operations personnel.*

***SOCC** is Akamai's Security Operations Center. DDoS protection experts in a security operations center customize defenses and protect customers against DDoS attack types observed as the attack happens.*

Exhibit P

Key concepts and terms

There are a few basic concepts you should know before you begin setting up your properties. This chapter provides a road map of all the terms you deal with when interacting with the Property Manager application.

How users access your content

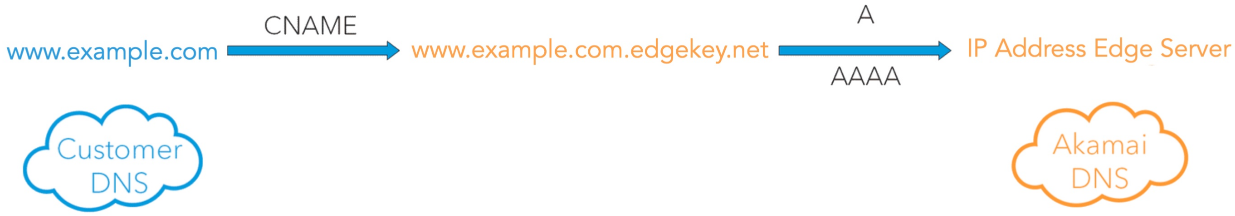
To understand the content delivery flow better, here are a few definitions you want to keep straight:

- **Domain Name Server (DNS)**. Keeps track of domain names and translates them to an IP address.
- **DNS CNAME**. A record that specifies that a domain is an alias for another domain. Configure your DNS server so that requests for each hostname resolve to a corresponding edge hostname, the latter referred to as the *canonical name*. In turn, servers across the Akamai network use their own CNAME records to resolve these edge hostnames to more specific local server names and ultimately their IP addresses.
- **Origin server**. The server you publish your content to.
- **Origin hostname**. A new DNS record that points to your origin server. You create this DNS record (for example, `origin-www.example.com`). See Types of hostnames for more details.
- **Edge server**. The Akamai network server.
- **Edge hostname**. The DNS record that points to the most optimal Akamai edge server. We provide this (for example, `www.example.com.edgesuite.net`). See Types of hostnames for more details.

How content delivery works

When users visit your site, their browsers do a DNS lookup of your domain name. Without Akamai, your DNS configuration returns an **A** or **AAAA** record that points to your origin server.

To serve your traffic through Akamai, you need to direct end users to an Akamai server. First, you change the DNS configuration for your domain. Instead of an **A** record, you need a **CNAME** record that redirects the DNS request to the edge hostname. Once the **CNAME** record is in place, Akamai DNS servers return the IP address of the best Akamai servers for the user's request. The browser uses the Akamai IP addresses to retrieve content from your site. If necessary, the Akamai servers will look up your origin hostname to connect to your origin server to retrieve content.



Types of hostnames

There are three types of hostnames involved in serving your content through Akamai:

- **Origin server hostname**. A label you define for your physical server when you first establish it and connect to a computer network. Today your end users reach your origin server using your primary domain. After you go live on Akamai, your primary domain will point to the edge servers, so you need to create a new hostname for your origin (for example, `origin-www.example.com`). This new name will appear in the origin DNS record and tell our edge servers from where they can retrieve content to serve over the Akamai network.

- **Property hostname**. The name of your site or application that your users access by clicking a link or typing a URL in their browser, without the "http(s)://". Property hostnames are fully qualified domain names (FQDN) that consist of a subdomain indicating a property type (such as `www` , `m` for mobile, `img` for images, `shop` for an online store, and so on), and your main hostname (such as `example.com`). When you set up your property, you associate its hostname with an edge one. This way, the requests for your content are redirected to Akamai edge servers. The property hostname helps the edge network determine that the request belongs to you as our customer and apply a dedicated configuration file. A hostname plays two roles:

  - As an address: used to locate a server that can serve the site's content.

  - As a Host HTTP Header: used to tell the server which site the request is asking for. It is entirely possible for multiple sites to be served by the same server.

- **Edge hostname**. A CNAME target. That means, if your user requests a property hostname of `www.example.com` , their request is rerouted to Akamai edge hostname, for example, `www.example.com.edgekey.net` . This essentially gets your property onto the Akamai network. Once the DNS for a site points to an edge hostname, Akamai's mapping algorithms determine the IP address of the most optimal edge server. The edge server then accesses the property configuration file to deliver the content according to the rules you set. For example, how long to cache certain objects or what cookie type to send with the response to the client.

Your edge hostname settings may vary depending on the level of security you need for your traffic.

Content structure

You might want to use Akamai products with multiple digital assets. To ensure it's easy to manage them all, Akamai has developed a special content structure.

The list starts with the largest entity that is further subdivided into smaller content elements:

- **Accounts**. You can access all your services with an account key. While administrators may have access to more than one account, in general, they provision all their web assets under a single account within Akamai Control Center. Accounts can comprise several groups.

- **Groups**. Each account features a hierarchy of *groups*, which control access to properties and help consolidate reporting functions, typically mapping to an organizational structure. Using Identity and Access Management, account administrators can assign properties to specific groups, each with its own set of users and accompanying roles. Your access to any given property depends on the role set for you in its group.

- **Contracts**. Each account features one or more contracts, each of which has a fixed term of service during which specified Akamai products and modules are active.

- **Products**. Each contract enables one or more products, each of which allows you to deploy web properties on the Akamai edge network and receive associated support from Akamai Professional Services. Products allow you to create new properties, CP codes, and edge hostnames. They also determine the baseline set of a property's rule behaviors.

- **Modules**. Modules are add-ons to products that may enable additional rule behaviors. Different products support different sets of modules. Your ability to specify any given rule behavior depends on the currently active product and associated modules.
  API only: PAPI doesn't provide information directly about your selected modules, but it does allow you to determine the currently available behaviors and criteria they enable.

- **Properties**. A property is the most granular object in the hierarchy and the first step on your journey to deliver your content over Akamai. You can think of a property as a container for your Akamai setup and services for your site, application, download files, or streams.

Properties

The edge network caches your web assets near to servers that request them. A property, sometimes also referred to as a configuration, provides the main way to control how edge servers respond to various kinds of requests for those assets. Properties apply rules to a set of hostnames, and you can only apply one property at a time to any given hostname. Each property is assigned to a product, which determines which rule behaviors you can use.

Let's clear up the concepts you should be familiar with while editing properties:

- **Versions**. Each property has snapshot versions, which allow you to modify one instance of a property while another is activated. You can freely modify a property version, along with its component hostnames and rules, up until you activate it. Following activation, you need to create a new version to modify it further. You don't need to create a new property version for each modification you make. Versions use an ascending number as an ID. They also feature a timestamp, the username of the person who last modified it, and some notes.

- **Activations**. Once you're satisfied with any version of a property, an activation deploys it, either to the Akamai staging or production network. You activate a specific version, but the same version can be activated separately more than once. You can either cancel an activation shortly after requesting it, or in many cases, use a fast fallback feature within a matter of seconds to roll back a live activation to the previously activated version.
- **Metadata**. Once activated, the property settings are distributed to the Akamai network as an XML configuration file, also known as metadata. This low-level XML format combines information about the property's rules and hostnames.
- **Errors and warnings**. When you modify different aspects of a property, errors or warnings may pop up along the way. An error prevents you from activating a property version, but you can activate versions that yield less severe warnings. You may need to acknowledge each warning when you activate a version, but there's an option to skip them.

Configuration elements

Rules, matches, and behaviors are the heart of your property configuration. These elements let you decide how your content is delivered to the end users.

- **Rules**. Properties allow you to design rules to respond to different kinds of requests. A rule consists of behaviors and optionally a set of match criteria. In addition to a top-level default rule that determines overall behavior, you may include any number of your own rules, arranged in a tree structure up to five levels deep. You can control the order in which they apply and create parent-child relationships between rules.
  API only: PAPI provides an interface to the entire rule tree, not to each component rule.

  See Rules for more details.
- **Rule templates**. A template is a predefined rule that you can insert into your configuration as-is or modify to a slight extent. Rule templates also group together commonly used functionalities so that they are easy to find.
  API only: PAPI doesn't support the rule templates feature available in the Property Manager interface, since programming tools allow you to build your own rule templating system more flexibly.
- **Matches**. Match criteria are the conditions that help you to define when a behavior should be applied. They are the IFs in your configuration, for example, "If there's an origin failure like a timeout...".

  See Matches for more details.
- **Behaviors**. Also referred to as features, behaviors instruct the edge servers how to manage requests. They are the THENs in your configuration, for example, "... redirect user requests to an alternate hostname".

  See Behaviors for more details.
- **Advanced and custom behaviors**. Most behaviors translate to edge metadata in clearly prescribed ways. Functionality that falls outside what these predefined behaviors and criteria can do may be implemented as advanced metadata.

  Custom behaviors let you easily reuse advanced metadata in Property Manager configurations across your account.

> ### Advanced and custom features
> When implementing advanced and custom features, reach out to Akamai Professional Services to configure them for you.

- **Variables**. Many behavior and criteria values allow you to add a variable text that interprets at runtime on edge servers, typically based on details about the client request.

  See Variable overview for more details.

- **CP codes**. You need a content provider (CP) code to track any web traffic handled by Akamai servers. Akamai provides a CP code when you purchase a product, and you need it to activate any associated properties. You can generate additional CP codes, typically to implement more detailed billing and reporting functions, and to assign to customized properties. Each property requires at least one CP code behavior as part of its default rule.

- **API only**: **Schemas**. Some errors may result when a request isn't in the expected format. In that case, errors reference *request schemas* that represent the expected structure formatted as a JSON Schema object.

- **API only**: **Rule formats**. Akamai often modifies PAPI features, each time deploying a new internal version of the feature. By default, the Property Manager interface in Control Center uses the `latest` available feature versions and you may be prompted to upgrade your configuration. In the interest of stability, PAPI does not support this system of selective updates for each feature. Instead, PAPI's rule objects are simply versioned as a whole. These versions, which update infrequently, are known as *rule formats*.

  You use rule formats to freeze or update the versioned set of behaviors and criteria a rule tree invokes. PAPI users should assign the most recent dated rule format to freeze the set of features. Otherwise, if you assign the `latest` rule format, features update automatically to their most recent version. This may abruptly result in errors if JSON objects your rules specify no longer comply with the most recent feature's set of requirements. PAPI provides a more stable path to update rule formats that fixes these requirements for you.

  PAPI tracks rule formats in a database keyed by *rule format version* date strings. These reference *rule format schema* objects, which specify the full set of behaviors and criteria available for a given product and the set of modules it may enable, as well as their allowed options and option values.

- **API only**: **Client settings**. This API resource collects various configuration parameters for clients keyed to an authorization token.

API Only: Bulk operations

PAPI supports a set *bulk* operations to manage many properties' rule trees at once. See the Bulk Search and Update section for details on this feature.

- **Bulk searches.** Bulk searches use a flexible JSON path-based query syntax to search across all your activated property versions' rule trees. After your search request, search results become available asynchronously for all matching property versions. They include JSON path expressions that locate all the rule tree behaviors and criteria you searched on, for use in a subsequent bulk patch operation.

- **Bulk versions**. You feed the results of a bulk search into another operation that creates new versions for the entire set of properties. This new set of versions becomes available asynchronously.
- **Bulk patches**. In a bulk patch operation, you use the JSON path locators in conjunction with JSON Patch operators to form a set of customized instructions to modify each property's rule tree. The results become available asynchronously.
- **Bulk activations**. After batch-updating a set of properties, you can asynchronously bulk activate them to the staging or production network.

Updated 9 months ago

← Welcome to Property Manager

Workflow overview →

Did this page help you? **Yes** **No**

Exhibit Q

Akamai

MSOD: Stream Packaging User Guide

March 03, 2022

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Contents

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Welcome to Media Services On Demand: Stream Packaging

Media Services On Demand (MSOD): Stream Packaging allows use of various on demand file formats, such as FLV and MP4, with RTMP ingress and egress in either Adobe HDS, Apple® HLS, or progressive FLV delivery (HD Flash 1.0) outputs.

ⓘ **Note:** HD Flash 1.0 is only supported for existing customers of this product.

MSOD: Stream Packaging uses dynamic streaming to provide a high-quality video experience for end-users. It also offers support for the broadest range of video end points to maximize content reach.

Dynamic streaming is an enabling technology that automatically selects and streams the highest quality bit rate to a video end point based on connection speed and device capabilities. If the end user's connection speed changes, the player responds by requesting a different bit rate from MSOD and a seamless bit rate switch occurs.

For example, if the available bandwidth on an end user's connection drops from 2 Mbps to 1 Mbps, the player detects the change and requests the stream of a lower bit rate to avoid rebuffering or video stutter. If the connection speed increases, and the available bandwidth returns to 2 Mbps, the player asks MSOD to resume streaming at the original 2 Mbps rate.

All of this complexity is hidden from the end user and the result is a consistent, high-quality viewing experience.

Encoding guidelines

For general encoding guidelines, refer to the On Demand Encoding Best Practices document, available on Control Center.

Supported codecs/containers

HDS/HD Flash 1.0 outputs

The following table lists the codecs and containers that are supported by Adobe HTTP Dynamic Streaming and HD Flash 1.0.

Container	Video codec	Audio codec	Comments
FLV	H.264 VP6 Sorenson Spark (H.263)	MP3[1] AAC PCM Nellymoser	Video-only works as well. For Nellymoser, only the 8 khz and 16 khz mono sound formats are supported

[1] MP3 audio format is supported only if wrapped in a MP4 container (applies to HD Flash 1.0 and Adobe HTTP Dynamic Streaming).

Container	Video codec	Audio codec	Comments
F4V	H.264	AAC	
MP4	H.264	AAC MP3	Audio-only (AAC) or Video-only also works
F4F/F4M	H.264 VP6	AAC MP3	

HLS output

The following table lists the codecs and containers that are supported by Apple HTTP Streaming.

Container	Video codec	Audio codec	Comments
F4V/MP4	H.264 H.264 Baseline Profile Level 3.0[2] Main Profile Level 3.1[3]	AAC-LDC up to 48 kHz, stereo audio	Audio-only (AAC) or Video-only also works

Following are links to web pages containing the video playback specifications for Apple iPad®, Apple iPhone®, and Apple iPod touch® devices, which list the types of encoding each device supports.

iPad	*http://www.apple.com/ipad/specs/*
iPhone	3G—*http://support.apple.com/kb/SP495* 3GS—*http://support.apple.com/kb/SP565* 4G—*http://www.apple.com/iphone/specs.html*
iPod Touch	2nd generation—*http://support.apple.com/kb/sp496* 3rd generation—*http://support.apple.com/kb/SP570* 4th generation—*http://support.apple.com/kb/SP594*

For more Apple HTTP Streaming encoding recommendations, refer to Apple's web page Best Practices for Creating and Deploying HD Live Streaming Media for the iPhone and iPad (*http://developer.apple.com/library/ios/#technotes/tn2224/_index.html*). Additionally, Apple has a web page with links to streaming documentation, including the latest best practice recommendations for encoding, deployment, app development, and app submission at *https://developer.apple.com/resources/http-streaming/*.

[2] iPhone and iPod Touch
[3] iPhone and iPad

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Origin server requirements

If you host your videos on your own server, you must have adequate resources provisioned so that servers can retrieve the videos. These are the requirements for origin servers:

- Servers must be HTTP 1.1 compliant to support byte-range GET requests.
- Servers must return a Last-Modified header on every byte range GET request; the header must be identical for all chunks of a given file and must not change unless the file is replaced.

  ⓘ **Note:** If your origin consists of multiple servers behind a load balancer, or if the origin host name returns multiple server IP addresses, you must ensure that, for a given file, the Last-Modified header is identical on each server and IP address.

- Servers must return an updated **Last-Modified** header when a file is replaced. You must also purge the file from the Edge servers to ensure that they fetch the new version.
- Servers must have adequate processing power and provisioned bandwidth to handle the maximum expected simultaneous requests from the servers

  ⓘ **Note:** This will never exceed, and is usually substantially less than, the maximum number of requests received when not using Media Services On Demand.

You need not maintain your own origin server if you use Akamai NetStorage.

Open issues

The Speex audio compression format is not supported.

Truncated FLV files are not currently handled.

I-frame playlists are not supported in dynamic stream packaging.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Create configurations

Before you can create new streams, you must create at least one stream packaging configuration to establish parameters for packaged streams.

You can create as many configurations as you have CP codes available, depending on how you would like to have your streams being reported on and billed.

1. Access https://control.akamai.com/.
2. Log in using an account that's been provisioned for access to MSOD Stream Packaging.
3. Open the application. Go to ☰ > **MEDIA** > **Other Media services** > **On-Demand Stream Packaging configurations**.

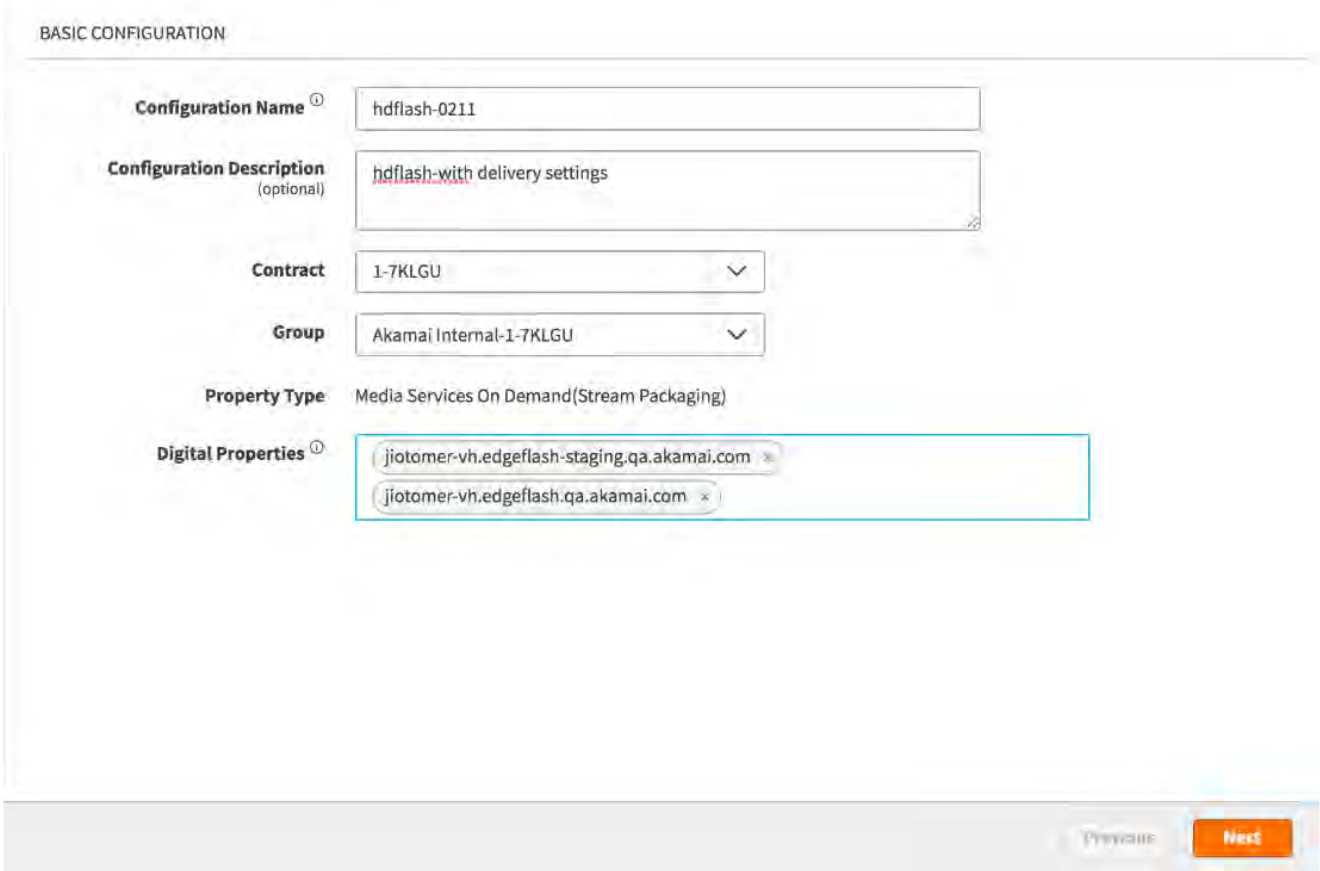
What's next?

Begin the creation process and *Specify basic settings* on page 5.

Specify basic settings

Click **Add Configuration** on the MSOD stream packaging configurations page to display the basic settings page.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf



BASIC CONFIGURATION

Configuration Name: hdflash-0211

Configuration Description (optional): hdflash-with delivery settings

Contract: 1-7KLGU

Group: Akamai Internal-1-7KLGU

Property Type: Media Services On Demand(Stream Packaging)

Digital Properties: jiotomer-vh.edgeflash-staging.qa.akamai.com, jiotomer-vh.edgeflash.qa.akamai.com

1. In the Configuration Name text box, enter a name for the configuration file.The configuration file is stored as an XML file, and the name you choose should typically reflect the digital properties it includes (for example, streampackaging.example.com.xml).

2. Optionally enter a description for your configuration.

3. In the **Digital Properties** text box, enter the domain name or hostname for the configuration. For example, **streampackaging.example.com**.

   You must include a hostname in the form of ***-vh.akamaihd.net**. For example, **customername-vh.akamaihd.net**. This will be created behind the scenes when you complete your configuration creation. Make sure that there is one and only one hostname ending in **-vh.akamaihd.net**.

   Staging digital properties for your **akamaihd.net** hostname will be automatically added and can be tested by activating configuration on the Staging network. For example, if your hostname is **customer-vh.akamiahd.net** then the corresponding staging hostname would be **customer-vh.akamiahd-staging.net**.

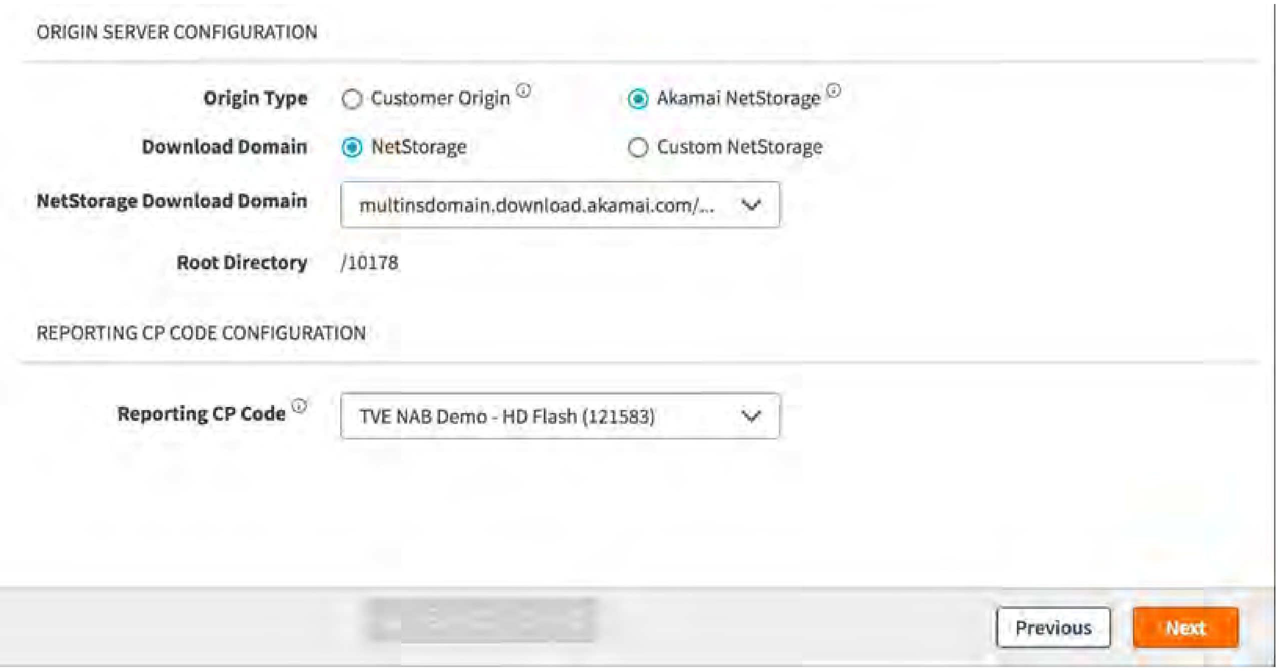
   ⓘ **Note:** The **-vh** suffix indicates it is associated with Media Services On Demand Stream Packaging, for example, **example-vh.akamaihd.net**. This suffix supersedes the previously-used **-f** and **-i** suffixes. However, if you have older configurations that use either of these, they will continue to operate normally.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

If you have more than one digital property, you can add additional digital properties clicking **+Add Another Additional Digital Property**. The limit is 10. If you need more digital properties, contact your Account Representative.

Specify origin and CP code

Click **Next** on the basic settings page to display the origin and CP code page.



ORIGIN SERVER CONFIGURATION

Origin Type	○ Customer Origin	● Akamai NetStorage
Download Domain	● NetStorage	○ Custom NetStorage
NetStorage Download Domain	multinsdomain.download.akamai.com/...	
Root Directory	/10178	

REPORTING CP CODE CONFIGURATION

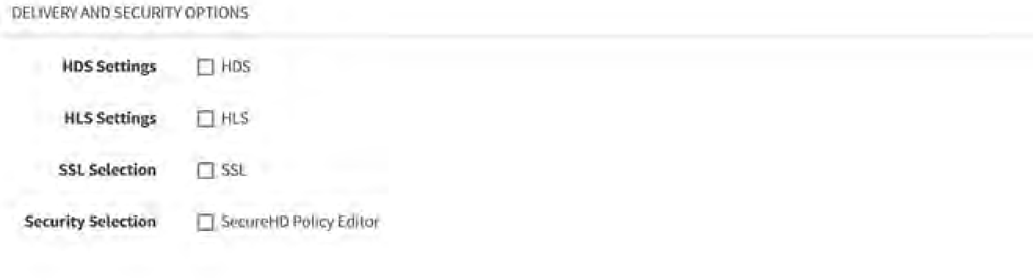
Reporting CP Code	TVE NAB Demo - HD Flash (121583)

Under origin server configuration, specify the following:

1. Select **Customer Origin** or **Akamai NetStorage** as the origin type, depending on what you plan to use as an origin for your on demand content.
2. Select **NetStorage** as the download domain for Akamai NetStorage origin type. Select **Custom NetStorage** for Customer Origin. See *Origin server requirements* on page 4 for information on hosting your own content.

Specify delivery and security options

Click **Next** on the origin and CP code page to display the delivery and security options page.



DELIVERY AND SECURITY OPTIONS

HDS Settings □ HDS

HLS Settings □ HLS

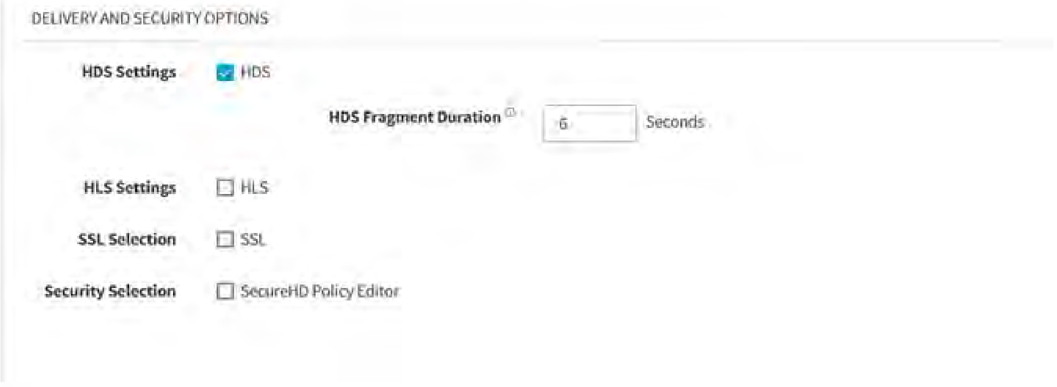
SSL Selection □ SSL

Security Selection □ SecureHD Policy Editor

You can configure HDS settings, HLS settings, SSL selection, and security selection on this page.

Configure HDS settings

Select HDS settings in the delivery and security options page.



DELIVERY AND SECURITY OPTIONS

HDS Settings ☑ HDS

HDS Fragment Duration 6 Seconds

HLS Settings □ HLS

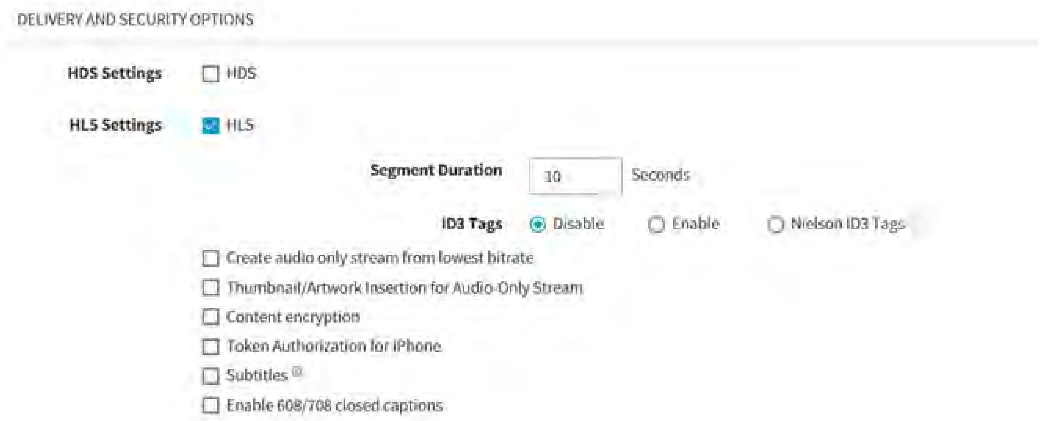
SSL Selection □ SSL

Security Selection □ SecureHD Policy Editor

Specify a value (in seconds) for the HDS Fragment Duration. This value should be equal to the keyframe interval or multiples of it. The default value is six (6) seconds, which would work for one-, two-, three-, and six-second keyframe intervals. If your interval is different from these values, set the appropriate value in this field. Setting this value incorrectly could cause undesirable bit rate switch downs in HDCore versions lower than 2.8.

Configure HLS settings

Select HLS Settings in the delivery and security options page to display the HLS settings options.



DELIVERY AND SECURITY OPTIONS

HDS Settings □ HDS

HLS Settings ☑ HLS

Segment Duration 10 Seconds

ID3 Tags ● Disable ○ Enable ○ Nielson ID3 Tags

□ Create audio only stream from lowest bitrate

□ Thumbnail/Artwork Insertion for Audio-Only Stream

□ Content encryption

□ Token Authorization for iPhone

□ Subtitles

□ Enable 608/708 closed captions

Specify segment duration

In the Segment Duration field, enter a duration (between 2 and 60 seconds) for your segments or fragments. The default duration is 10 seconds. If you change the HLS segment duration for an existing VOD stream packaging configuration, ensure that you purge any content from the cache.

Specify ID3 tags

ID3 tags are disabled by default.

Select one of the following:

- Click **Disable** if you don't want to specify any ID3 tags.
- Click **Enable** to optionally activate timed metadata using options in the ID3 Tags area.Timed metadata is custom metadata that is added to the source content for passing the following: Cue points, timecodes, ad markers, an album name, a title — ID3 tags.
- Click **Nielsen ID3 Tags** to enable program ratings using Nielsen ID3 tags. For details about Nielsen ID3 tags (rating information), see *http://www.nielsen.com*.

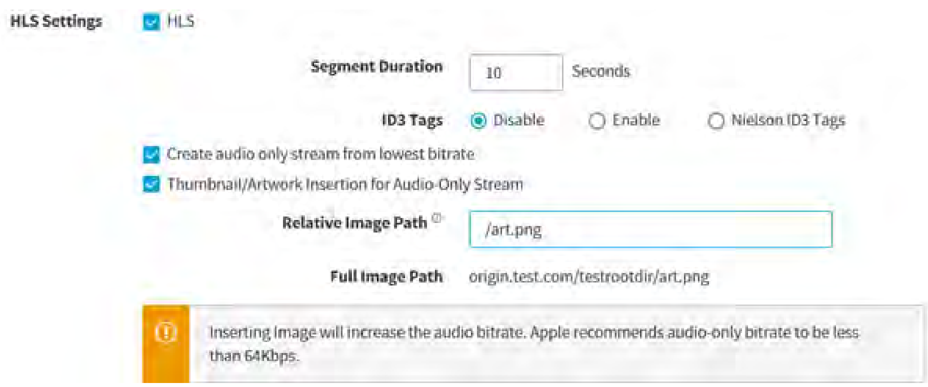
Create audio only stream from lowest bitrate

If you want automatically extract an audio-only stream from your lowest published bit rate, select **Create an audio only stream from lowest bitrate**. If you are publishing an audio-only stream, and you select this option, your stream takes precedence over that produced by Media Services On Demand.

Thumbnail/Artwork insertion for audio only stream

Select **Thumbnail/Artwork insertion for audio only stream** to to display an image when the audio-only stream is played.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf



HLS Settings ☑ HLS

Segment Duration 10 Seconds

ID3 Tags ● Disable ○ Enable ○ Nielson ID3 Tags

☑ Create audio only stream from lowest bitrate

☑ Thumbnail/Artwork Insertion for Audio-Only Stream

Relative Image Path /art.png

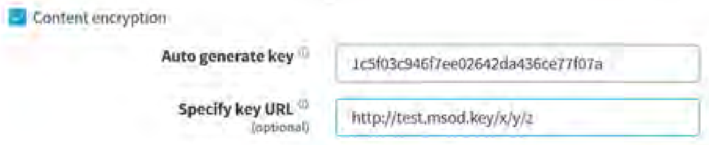
Full Image Path origin.test.com/testrootdir/art.png

Inserting Image will increase the audio bitrate. Apple recommends audio-only bitrate to be less than 64Kbps.

Enter the path of the image that you would like to display in the Relative Image Path field. MSOD supports only jpg and png format files of 40 KB or less.

Configure content encryption

Select Content encryption to deliver encrypted content from the edge servers to the player runtime. Encryption uses AES-128 bit crypto algorithm per Apple's specification.



☑ Content encryption

Auto generate key 1c5f03c946f7ee02642da436ce77f07a

Specify key URL (optional) http://test.msod.key/x/y/z

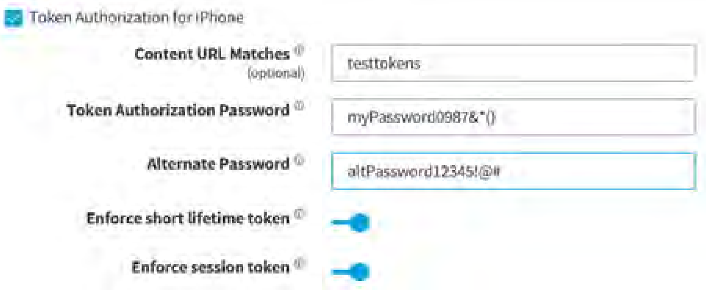
Note: You cannot use content encryption along with the secure HD policy editor.

The encryption key in 32 hex digits is automatically filled in the Auto generate key field.

Enter the encryption key URL to provide to clients in the playlist file. For example: `http://foo.bar.baz/x/y/z`

Configure token authorization for iPhone

Select **Token Authorization for iPhone** to enable token-based authentication for iPhones. This authenticates the users who attempt to log in to a server, a network, or some other secure system, using a security token provided by the server.



☑ Token Authorization for iPhone

Content URL Matches (optional) testtokens

Token Authorization Password myPassword0987&*()

Alternate Password altPassword12345!@#

Enforce short lifetime token ●

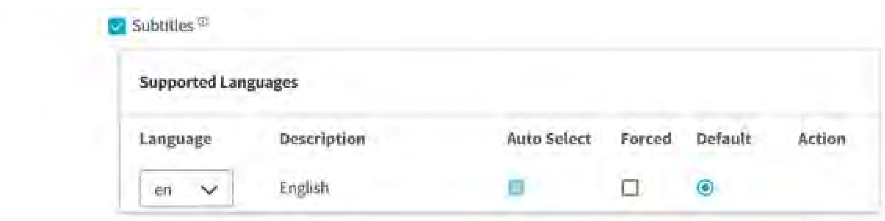
Enforce session token ●

Enter the following:

1. Enter a string to match the requested URL against when using token authorization. Use * to specify wildcard matches. For example, if you enter `/protected/*`, only content in this folder is protected by token authorization. If left blank, all content will have token authorization enforced.
2. Enter a long and complex string as the token authorization password.
3. Enter a long and complex string as the alternate password, which is used temporarily when you change the token authorization password.
4. Use the slider to (optionally) select enforce short lifetime token on initial player requests.
5. Use the slider to (optionally) select enforce session token for all ongoing requests, including playlists, content segments, and keys.

Configure subtitles

Select **Subtitles** to configure subtitles in your HLS stream.



☑ Subtitles

Supported Languages

Language	Description	Auto Select	Forced	Default	Action
en	English	☑	□	●	

There should be either DFXP or WebVTT file per language, hosted in the same storage location as the media asset, if in a different location should be specified with the appropriate query string.

File should be named as per the naming convention consisting of language code and media identifier, along with underscore '_' separator. Default media identifier will be the same as the media asset name, Media identifier can be overridden in a query string. If the file name follows different naming convention then the complete file name will be specified in a query string.

1. Select a language for the subtitles. The description automatically appears.
2. Select Auto Select or Forced.

Use a DFXP/WebVTT file per each language that needs to be located in the same storage as your media asset. If it is stored in a different location, you must use query strings to set your subtitles as described in the section *Examples of setting up Subtitles via UI* on page 24.

If you are setting up your subtitles on UI, use the following functions:

- Select the input file format (the **VTT** or **DFXP** radio button) and file naming convention (the **Prefix Identifier** or **Suffix Identifier** radio button). The names of the DFXP/WebVTT file should consist of a media asset and a corresponding language code separated by an underscore.

  You can change the file naming convention in some of your output during playback using the query strings as described in the section *Query strings for setting up subtitles* on page 26.

- Enable the default for subtitles' language and select this default using the scroll-down menu and the **DEFAULT** radio button. Only one default language can be allowed. The subtitles in the default language start on playback. After you select your default language, it will automatically be added to the list of the autoselected languages.

- Add another language for your subtitles using the **Add Another** option and the **LANGUAGE CODE** scroll-down menu.
- Add a customer language (which is not on the preset options of the **LANGUAGE CODE** scroll-down menu) for your subtitles using the **Add Customer Language** option.

ⓘ **Note:** You can have up to 15 languages on the list. As soon as you add the fourth language, the **Add Another** and **Add Customer Language** options disappear from UI.

- If the **AUTOSELECT** option is turned on for a certain language, and the **Auto (Recommended)** option is selected during playback, this language appears as an available subtitle choice during playback.
- If the **FORCED** option is selected, this language is automatically turned on for your video asset.

ⓘ **Note:** Some players do not honer the **FORCED** option.

For examples that illustrate how the enabled **DEFAULT**, **AUTOSELECT**, and **FORCED** options affect your playback, refer to *Examples of setting up Subtitles via UI* on page 24.

Use closed captioning

To enable closed captioning, select **Enable 608/709 closed captions**.

Closed captions is a process that allows insertion of data or text that corresponds to the video scene. This can be done to provide accessibility to hearing-impaired persons (close captions or open captions), language translations for multi-region programming, or any other reason a producer chooses.

You can pass closed captions to a viewer in one of these ways:

- Embedded into the video file itself.
- Delivered in an accompanying *sidecar* file.

ⓘ **Note:** In the h.264 input stream, if the subtitles text is present in the NALU header type 6 (SEI) as ATSC picture user data, it will be passed through and available in the HLS output.

Select SSL

Before you configure SSL, ensure that the origin supports SSL. Select **SSL** to configure end-to-end SSL from edge to origin. This prevents protocol downgrades throughout the path and mixed content warnings.

Activate your configuration

The Edge Staging Network (ESN) provides an environment to test your configurations without impact on your live, production traffic. After successful testing on ESN, activate the same configuration file version for production. Most standard current-version production features can be tested on ESN, but note that ESN is for testing functionality and not load, stress, or performance.

Contact your Akamai representative to review your configuration for any of the following incompatible features:

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Global Traffic Manager GTM (*.akadns.net) Two Tree configurations SiteShield Additional information and training is available in the ESN User Guide and the On Demand Training Module.

Akamai Staging hostnames (**customer-vh.akamaihd-staging.net**) corresponding to your hostname (**customer-vh.akamaihd.net**) will be added for all staging activations for Universal Streaming. You can test these changes on Akamai Staging networkk before activating the configuration on production.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Set up the video playback for HDS and HD Flash 1.0 outputs

Media Services On Demand playback applications for HDS and HD Flash 1.0 outputs are built using provided Adobe ActionScript® 3.0 code library called **HDCore**. The purpose of this library is to make it easy for developers to add Media Services On Demand functionality to existing players.

You can build players for Media Services On Demand: Stream Packaging using any IDE that allows you to author in ActionScript 3.0 and compile to Flash Player 10. The core library is pure ActionScript 3.0 and is not dependent on the Adobe Flex® framework or any other framework for implementation. The most common development environments are Adobe Flash Builder® and Adobe Flash Designer.

Media Services On Demand delivers HD Flash 1.0 FLV files progressively to a Flash Player, version 10.0 or greater. Progressively here means over a network connection and through a null **NetConnection**. Contrary to the restrictions most associated with progressive delivery, you can instantly seek any point in these files, smoothly switch between different renditions of the content at alternate bit rates, and play On Demand content. These FLV files can actually wrap H.264 content, and in fact, the majority of content delivered over the network is H.264.

Helpful definitions for HDS output

This section introduces you to some key terms and definitions.

File type	Description	Extension	Example
Flash Media Manifest File	This file is requested by the player once at the start of the play session. It contains a list of all the available bit rates (tracks) for the requested stream along with the DVR window and other related metadata. The player uses the bit rates in this file to make switching decisions based on available bandwidth.	.f4m	manifest.f4m
Fragment File	These are small files suited for Media Services On Demand that contain the audio and/or video media data in the MP4 format.	N/A	1200_2d7ba93e14ed84fc-p_Seg1-Frag5456490

Use Media Services On Demand player components

As previously stated, Media Services On Demand playback applications for HDS and HD Flash 1.0 outputs are built using a Media Services On Demand provided ActionScript 3.0 code library called **HDCore**. The purpose of this library is to make it easy for developers to add Media Services On Demand functionality to existing players.

- **HDCore**

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

  - There are two core classes within this library:

    com.akamai.hd.ZStream — is used for HDS output.

    com.akamai.hd.HDNetStream — is used for HD Flash 1.0 output.

  - Full ASDOCS are available for each of these classes and are included with any code distribution. The code itself is distributed as a SWC library.

  - HDCore 2.3+ integrated into a framework built on Flash Player 10.1 and above is the minimum version required for HDS output.

  - Media Analytics integrated into HDCore:

    HDCore, version 1.1+ is the minimum version required. If you wish to use Media Analytics without integrating it into HDCore, you can use any HDCore version you like.

- **OSMF Plugin**

  - Advanced Streaming Plugin for OSMF is not required in order to use Adobe HTTP Dynamic Streaming. If you choose, however, to use the OSMF player and also leverage all Media Services On Demand features, this plugin provides support for it; version 1.0.4+ is the minimum version that supports it. For more details, refer to *Advanced Streaming Plugin for OSMF* available on Control Center.

    ⓘ **Note:** Usage of Player Components is enforced in all cases in which Adobe HTTP Dynamic Streaming is used.

Use other compatible playback methods

- LongTail Ad Solutions' JW Player® component

  For details, refer to *JW Player Plugin User's Guide*.

- Flowplayer, Ltd.'s flow player® video player component

  For details, refer to *Advanced Flowplayer Provider User's Guide*.

Use Media Services On Demand support players

Media Services On Demand: Stream Packaging provides support video players that you can use to test and troubleshoot your streams. Access them at the following locations:

- HDS output:

  HDS Test Player

- HD Flash 1.0 output:

  Flash HD Test Player

Serve your content

SMIL files and Client-Side URL syntax provide definitions to end-users' players regarding the multiple bit rates that are available for a given On Demand asset. Following are the situations in which you can use each, based on container type (these can be served from either NetStorage or your own Origin):

FLV	SMIL = YES Client-Side URL Syntax = YES
F4V/MP4	SMIL = YES Client-Side URL Syntax = YES
F4F	SMIL = NO Client-Side URL Syntax = NO *F4M = YES

*F4M is generated during the packaging process.

Using SMIL files

When streaming multiple bit rate assets, your Flash playback applications make use of SMIL files to map the available bit rate to the appropriate On Demand files. Following are some examples of SMIL files used with On Demand content.

Option 1: "vod" player feature

Use the "vod" feature in the player SMIL. This is the easiest and most foolproof way to do things and is almost always the right option.

example1a.smil:

```
<?xml version="1.0"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN" "http://www.w3.org/2001/
SMIL20/SMIL20.dtd">
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
<head>
<meta name="title" content="This Is My Stream" />
<meta name="HDBase" content="http://example.akamaihd.net/" />
<meta name="vod" content="true" />
</head>
<body>
<switch id="whatever">
<video src="video.300k.mp4" system-bitrate="300000"/>
<video src="video.500k.mp4" system-bitrate="500000"/>
<video src="video.800k.mp4" system-bitrate="800000"/>
<video src="video.1000k.mp4" system-bitrate="1000000"/>
</switch>
</body>
</smil>
```

Note that **HDBase** must include the protocol (**HD** or **HDs**) and the domain name, but no additional path components. Also, all subdirectories must be listed at the beginning of each URL (See example 1b).

Note: The configuration typically prepends certain path elements when requesting content from NetStorage such as the top-level CP code directory (for example, /9389). It is unnecessary, therefore, to explicitly add these path components in the URL used in the SMIL file.

All SMIL files referencing the same streams should make all of the bit rates available for maximum caching efficiency, since each stream is actually cached separately at the Edge if it is referenced from a different complete set. The HDCore classes provide a mechanism to limit the maximum bit rate that the player chooses if you do not want a given player session to have the option of switching up to the highest bit rate(s).

This example illustrates how to include a subdirectory in an On Demand SMIL file.

example1b.transformed.smil:

```
<?xml version="1.0"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN" "http://www.w3.org/2001/
SMIL20/SMIL20.dtd">
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
<head>
<meta name="title" content="This Is My Stream" />
<meta name="HDBase" content="http://example.akamaihd.net/" />
<meta name="vod" content="true" />
</head>
<body>
<switch id="whatever">
<video src="subdir/video.300k.mp4" system-bitrate="300000"/>
<video src="subdir/video.500k.mp4" system-bitrate="500000"/>
<video src="subdir/video.800k.mp4" system-bitrate="800000"/>
<video src="subdir/video.1000k.mp4" system-bitrate="1000000"/>
</switch>
</body>
</smil>
```

Note: The content paths here do not need to be identical except for the bit rate specification. However, the player's behavior is to construct a single URL containing all paths in the SMIL file, and the shorter the length of the common prefix and suffix across all of the paths, the shorter will be the resulting URL. An excessively long internal URL may exceed limits within the player, Flash runtime, browser, or Media Services On Demand, causing difficulty with playback.

Option 2: Server-side SMIL

You can also use a "server-side SMIL" to associate a set of bit rates with a short ".ssmil" URL, which is then referenced from a client-side SMIL.

Note: This option is much more error prone and complex, so option 1 is almost always preferred.

This example consists of `example2a.ssmil`, which is uploaded onto the origin as `/subdir/example2a.ssmil`, and `example2a.smil`, which is made available separately; the latter's URL is the one sent to the player.

example2a.ssmil

```
<?xml version="1.0"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN" "http://www.w3.org/2001/
SMIL20/SMIL20.dtd">
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
<head>
<meta name="title" content="This Is My Stream" />
<meta name="HDBase" content="/subdir/" />
</head>
<body>
<switch id="whatever">
<video src="video.300k.mp4" system-bitrate="300000"/>
<video src="video.500k.mp4" system-bitrate="500000"/>
<video src="video.800k.mp4" system-bitrate="800000"/>
<video src="video.1000k.mp4" system-bitrate="1000000"/>
</switch>
</body>
</smil>
```

example2a.smil

```
<?xml version="1.0"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN" "http://www.w3.org/2001/
SMIL20/SMIL20.dtd">
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
<head>
<meta name="title" content="This Is My Stream" />
<meta name="HDBase" content="http://example.akamaihd.net/" />
</head>
<body>
<switch id="whatever">
<video src="subdir/example2a.ssmil/bitrate=300000" system-bitrate="300000"/>
<video src="subdir/example2a.ssmil/bitrate=500000" system-bitrate="500000"/>
<video src="subdir/example2a.ssmil/bitrate=800000" system-bitrate="800000"/>
<video src="subdir/example2a.ssmil/bitrate=1000000" system-bitrate="1000000"/>
</switch>
</body>
</smil>
```

The ".ssmil" file sets `HDBase` as the object's path only. You can also set `HDBasemerri` to `/` and then repeat `/subdir/` in each video `src` tag in the .ssmil. The client-side ".smil" file still must include the `subdir` element as part of each individual video's `src` tag.

A server-side SMIL must have the extension ".smil" or ".ssmil" for the On Demand system to treat it as a SMIL file instead of a single-bit-rate video file.

The files in Example 2a can also be combined by renaming the `HDBase` and `src` in example2a.ssmil to `server-HDBase` and `server-src`, respectively, and then combining the `video` tags together. Note that the default configuration will not serve the ".smil" file to the end user from "example.akamaihd.net," so either the same file must be hosted in the On Demand origin location, as well as in another location, or a custom configuration must be crafted to serve the ".smil" files (but not other files) from the common origin location under a different hostname or something similar. Your Account Representative can assist in crafting this configuration if it is required.

example2b.smil:

```
<?xml version="1.0"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN" "http://www.w3.org/2001/
SMIL20/SMIL20.dtd">
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
<head>
<meta name="title" content="This Is My Stream" />
<meta name="HDBase" content="http://example.akamaihd.net/" />
<meta name="server-HDBase" content="/subdir/" />
</head>
<body>
<switch id="whatever">
<video src="subdir/example2b.smil/bitrate=300000" server-src="video.300k.mp4"
system-bitrate="300000"/>
<video src="subdir/example2b.smil/bitrate=500000" server-src="video.500k.mp4"
system-bitrate="500000"/>
<video src="subdir/example2b.smil/bitrate=800000" server-src="video.800k.mp4"
system-bitrate="800000"/>
<video src="subdir/example2b.smil/bitrate=1000000" server-
src="video.1000k.mp4" system-bitrate="1000000"/>
</switch>
</body>
</smil>
```

Using client-side URL syntax

If you are unable to generate or provide server-side SMIL files, you can alternatively use a client-side URL format with which you can encode the entire SMIL contents in a single URL. For Media Services On Demand (Stream Packaging), the client-side URL takes the form:

```
http://<config_name>-vh.akamaihd.net/z/<subdirectory>/
<common_filename_prefix>,
<bitrate>,<bitrate>,<bitrate>,<bitrate>,common_filename_suffix>.csmil/
manifest.f4m
```

So a published URL to the device for this form of integration might look like:

```
http://example-vh.akamaihd.net/z/movies/example2a_,
300000,500000,800000,1000000,_event1.mp4.csmil/manifest.f4m
```

Using the above examples and assuming a CP code of 9389, your content would reside on NetStorage at example.download.akamai.com/9389/movies/ as:

- example2a_300000_event1.mp4
- example2a_500000_event1.mp4
- example2a_800000_event1.mp4
- example2a_1000000_event1.mp4

Note: If you plan to stream just a single bit rate file, it does not need client-side URL and could be added by itself. For example, a single mp4 file called hello.mp4 can have a syntax of <hostname>/z/hello.mp4/manifest.f4m.

Use query strings (for HDS output only)

There is one query string available for use with your client-side URL for HDS output.

Note: The values below are in kbps and are applied to the combined audio + video bit rate.

Bit rate filtering (b)

If you would like to filter out certain bit rates — to test a particular bit rate version, for example — you can do so by adding a `b` query string to the end of the publish URL. There are four uses:

- Filter all bit rates except those specifically designated (and the lowest audio rate); the string takes the form:

  ```
  b=bitrate,bitrate,bitrate,...
  ```

  So, for example, if you had bit rates 200, 300, 700, and 1200, and you wanted to include only bit rates 200, 700, and the lowest audio rate, your URL would appear as:

  ```
  uri/manifest.f4m?b=200,700
  ```

  This would produce only the 200 and 700 bit rate streams, as well as the lowest audio bit rate.

- Designate a bit rate range (in kbps) outside of which the stream or streams you wish to filter fall; the string takes the form:

  ```
  b=lower_end-upper_end
  ```

  So, for example, if you had bit rates 200, 300, 700, and 1200, and you wanted to include all bit rates at or greater than 200 and at or less than 700, your URL would appear as:

  ```
  uri/manifest.f4m?b=200-700
  ```

  This would produce the 200, 300, and 700 bit rate streams.

- Designate the low end of a bit rate range (in kbps) where all bit rates below the value are filtered; the string takes the form:

  ```
  b=low_end-
  ```

  So, for example, if you had bit rates 200, 300, 700, and 1200, and you wanted to include all bit rates at and higher than 300 kbps, your URL would appear as:

  ```
  uri/manifest.f4m?b=300-
  ```

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

This would produce the 300, 700, and 1200 bit rate streams.

- Designate the high end of a bit rate range (in kbps) where all bit rates at and above the designation are filtered; the string takes the form:

  ```
  b=0-high_end
  ```

  So, for example, if you had bit rates 200, 300, 700, and 1200, and you wanted to include all bit rates lower than or equal to 700 kbps, your URL would appear as:

  ```
  uri/manifest.f4m?b=0-700
  ```

  This would produce the 200, 300, and 700 bit rate streams.

- Any combination of the above.

  So, for example, if you had bit rates 200, 300, 700, 1200, and 1800, and you wanted to include all bit rates except 700, your URL could appear as:

  ```
  uri/manifest.f4m?b=0-300,1200-
  ```

  This would produce only the 200, 300, 1200, and 1800 bit rate streams, as well as the lowest audio bit rate.

Flexible playback (start | end)

This query string enables end-users to begin playback at a specific time (in seconds) within the VoD stream. For example:

```
uri/manifest.f4m?start=30&end=500
```

Note: The start time is mandatory but the end time is not. If the end time is not provided, the stream will play until the end of the VoD asset.

Fragment duration (sd)

This query string can be used to override a stream’s fragment duration in case its keyframe interval and, hence, its fragment duration is different than what was set in the configuration (values should be designated as seconds (s) or milliseconds (ms). For example:

```
uri/manifest.f4m?sd=6006ms
```

```
uri/manifest.f4m?sd=6s
```

Test your content

You should test your content before making it available to your end-users to ensure a problem-free experience.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

ⓘ **Note:** Staging hostnames are added automatically for all **akamaihd.net** domains in order to test configurations on the Staging network. Staging hostnames will look as follows: **akamaihd-staging.net**.

Applying security

Security for HDS output

Security for HDS output is provided by the SecureHD Policy Editor (SPE), which is configurable through Control Center. For information regarding SPE and its configuration, refer to the *SecureHD Policy Editor User's Guide Network* available on Control Center.

ⓘ **Note:** It is recommended that you do not have more than 10 hostnames associated with SecureHD Policy Editor for performance reasons. If you require more than 10 hostnames associated with SecureHD Policy Editor, contact your Account Representative for assistance.

Security for HD Flash 1.0 output

The following are security offerings for HD Flash 1.0 streams delivered with Media Services On Demand: Stream Packaging. Each item is disabled in the default configuration, so they must be explicitly enabled by your Account Representative.

- *Using Player Verification* on page 22 — Verifies that the SWF attempting to play the content is valid and authorized. This feature is similar in functionality to SWF Verification offered by Adobe's Flash Media Server; it is, however, an entirely different approach to SWF authentication and does not use Flash Media Server, or require an RTMP connection to be activated. If the SWF fails to pass the authentication test, the Edge server stops delivering content to it after five (5) seconds.
- *Using Token Authorization* on page 23 — Protects the initial content request. A customer-built token service generates a unique, single-use, public, time-delimited token for the content, based on a secret shared with Media Services On Demand. This token service only generates the token if the user is a valid user for the application. Examples might include LDAP or a cookie for successfully logging in to a site. If the token is incorrect or missing, no content is returned at all from the Edge servers.
- **GEO Blocking** — Allows you to control which world regions are able to view your stream.

Each security type can be implemented independently of the others or together in any combination. The following examines how to implement each of these.

Using Player Verification

You must ensure that only SWFs you have authorized are allowed to connect to your streams.

How to

1. Inform your Account Representative that you wish to activate Player Verification on your Media Services On Demand account so that the server side is correctly provisioned. They will also provide you details for accessing the auth bin on NetStorage. If you are already using SWF Verification for your RTMP delivery over Flash Media Server, this is the same bin.

2. Upload the uncompressed version of the Flash SWF (meaning the binary SWF is uncompressed) that you wish to authorize for Media Services On Demand. When you export a SWF from Flex Builder™ or Flash Builder, it is compressed by default. In Flash CS4 and CS5, it is also compressed by default, although there is an option to save it as uncompressed. Either way, to make the process simpler for you, a simple uncompression application is available for you — *SWF Authentication Utility*.
3. This online application allows you to upload your standard compressed SWF. It uncompresses it and then allows you to download it to your desktop. You can then upload it to the auth bin described in step 1 above.
4. In the case of players that dynamically load sub-SWFs at run-time, the SWF you should use for auth is the one that controls (or establishes) the STAGE object in your player. So if **a.swf** loads **b.swf**, which uses HDNetStream, the SWF you should upload for Player Verification is **a.swf**—the one at the top of the display stack.
5. The last action you need to take to enable Player verification is to set the **displayObject** property on your HDNetStream instance by passing it a reference to any object that participates in the display stack and that has access to the **stage** property. Exactly which object you pass is unimportant. Since HDNetStream extends NetStream, it itself has no visibility into the player, and so you must provide it via this property. If you fail to set this property prior to calling **play()**, Player Verification will fail, even if you have uploaded the correct uncompressed SWF to your auth bin.

What you should see

Once Player Verification is enabled, the Edge server will start delivering content and then will query the Flash client to authenticate itself. This process will take about 1.5 seconds. If the Edge server cannot successfully authenticate the SWF within 5 seconds, it terminates the connection. If authentication is successful, playback will continue uninterrupted. Verification repeats itself with each new **play()** request the HDNetStream class makes.

Using Token Authorization

To enable Token Authorization protection for your player and content, take the following steps.

How to

1. Inform your Account Representative that you wish to activate Token Authorization protection for your HD Flash 1.0 content. This ensures the server side is correctly configured.
2. Your Account Representative will provide you sample server-side scripts that you run to generate a token service. These scripts are available in all the common server-side languages (ASP, Perl, PHP, and Sun Microsystems® Java™ languages). There will be a shared secret (and an optional salt) that both Media Services On Demand and your service code share.
3. The service should apply whatever business logic your application needs to validate a specific user. For example, it could require a session cookie to be present in the request, which would indicate that the user has authenticated themselves against a portal or login gateway. It could also carry an explicit token that you hand to the player upon instantiation. You could filter by IP address if you wish to authorize use solely within a company LAN, etc.
4. When the Web service is built, you must create an ActionScript class that calls it. This class must implement the ITokenService interface. Simply put, this interface requires the class to offer up a **requestTokenizedURL()** method, which the HDNetStream class uses to request a tokenized URL and then two callback functions to call on both success and error. The callback functions may seem a bit antiquated, but the inability of ActionScript 3.0 to support events in interfaces necessitates their use.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

5. In your player, you must then set the **requiresEdgeAuth** property to **true** before calling **play()**.
6. In the player, you must also instantiate an instance of this TokenService class and register it with the HDNetStream prior to the calling **play()**. This registration is done by setting the tokenService property to the token service instance you just created.
7. The first time you call **play()** for an asset, the HDNetStream class generates the exact URL request and then calls the **requestTokenizedURL()** method on your token service. Your token service class calls your token service and receives a token. If it is successful, it calls back the **callBackFunctionOnSuccess** function that the HDNetStream defined. If it has a problem, then it calls back the **callBackFunctionOnFailure** function.
8. Assuming success, the HDNetStream class hands the EdgeAuth token to the Edge server. If that token is validated, playback continues indefinitely. If the token is rejected, the Edge server disconnects the delivery session immediately.

   Token Authorization has this construct because authorization tokens might be required at any time during playback, not only at the beginning. This is because the Edge server has a short timeout period of only 120 seconds. If you pause the player for more than 120 seconds, the network connection times out. The HDNetStream detects this and makes a new **play()** request to continue playback. To do so, it needs to request fresh EdgeAuth tokens from your service.

How to use subtitles

ⓘ **Note:** The Subtitles feature is a Beta version. Contact your Account Representative for access to this feature.

Subtitling is a process of overlaying video with text. This is done to provide accessibility to hearing-impaired people, language translation, or any other reason a producer chooses. Subtitles can be passed to a viewer in an accompanying *sidecar* file.

The following file formats supported for sidecar files:

- WebVTT — Web Video Text Tracks Format — Sidecar file format specified by W3C at *http://dev.w3.org/html5/webvtt/*.
- DFXP — Distribution Format Exchange Profile – Sidecar file format specified by W3C at *http://www.w3.org/TR/2006/CR-ttaf1-dfxp-20061116/*.

You need a separate sidecar file for each language.

If multiple subtitles are configured, all subtitle files must be present or no subtitles will be displayed.

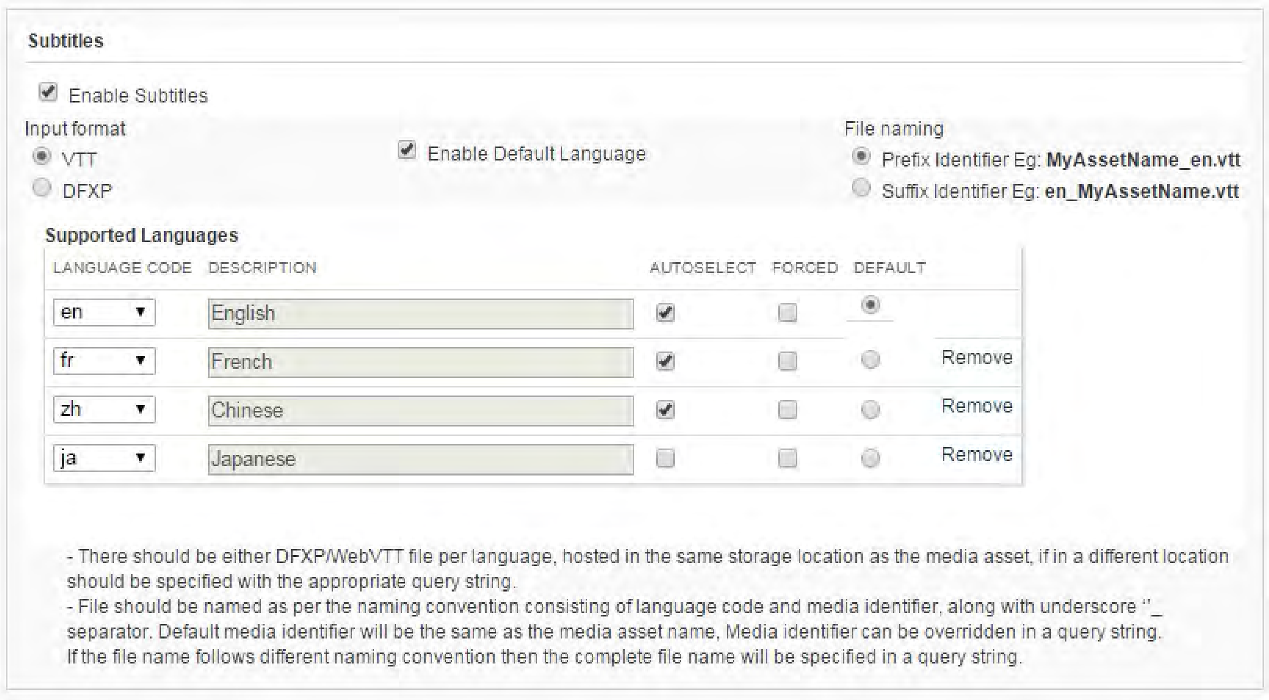
If you store your media assets and accompanying subtitles' files in the same location, you set up subtitles via the UI when creating your configuration. For information on subtitles embedded into the video file itself, see *Using closed captions* on page 29.

Examples of setting up Subtitles via UI

The Subtitles feature is a Beta version. Contact your Account Representative for access to this feature.

To illustrate how the enabled DEFAULT and AUTOSELECT options affect your playback, let us use the example shown in the following figure.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf



Subtitles

☑ Enable Subtitles

Input format

● VTT

○ DFXP

☑ Enable Default Language

File naming

● Prefix Identifier Eg: **MyAssetName_en.vtt**

○ Suffix Identifier Eg: **en_MyAssetName.vtt**

Supported Languages

LANGUAGE CODE	DESCRIPTION	AUTOSELECT	FORCED	DEFAULT	
en	English	☑	□	●	
fr	French	☑	□	○	Remove
zh	Chinese	☑	□	○	Remove
ja	Japanese	□	□	○	Remove

- There should be either DFXP/WebVTT file per language, hosted in the same storage location as the media asset, if in a different location should be specified with the appropriate query string.
- File should be named as per the naming convention consisting of language code and media identifier, along with underscore "_" separator. Default media identifier will be the same as the media asset name, Media identifier can be overridden in a query string. If the file name follows different naming convention then the complete file name will be specified in a query string.

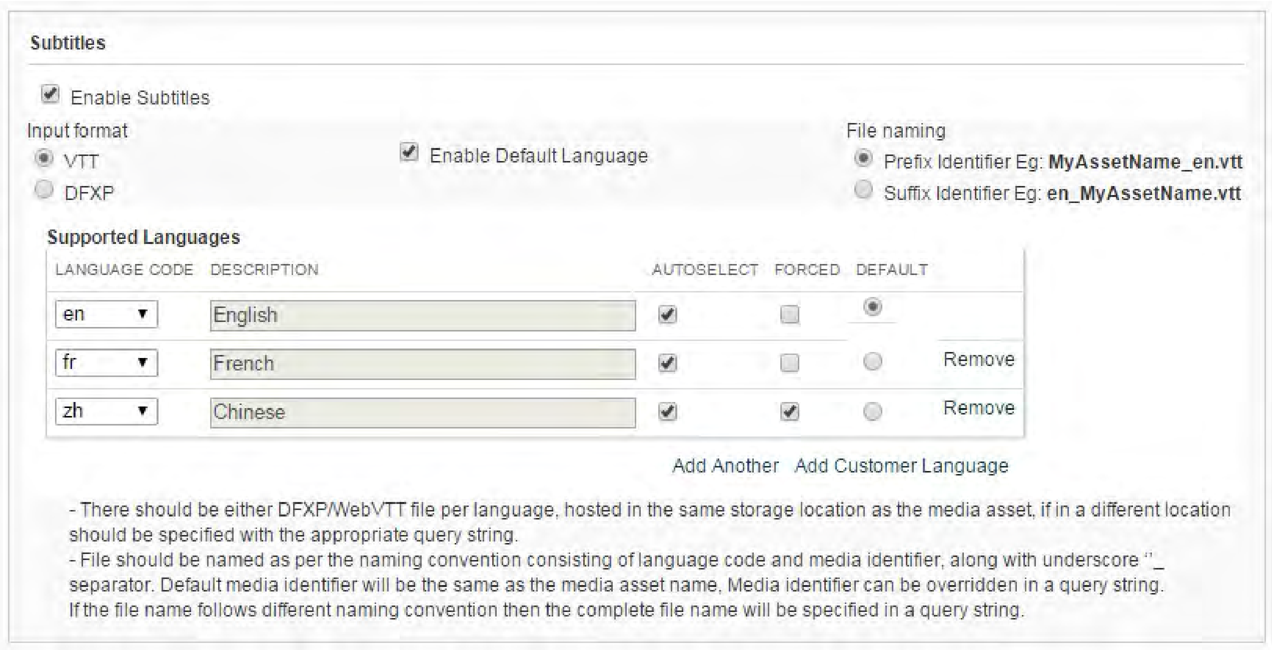
Example 1: Setting Up Subtitles During Provisioning on Control Center (Using AUTOSELECT and DEFAULT options)

In this example, our media asset for the movie could be accompanied by several subtitle files (English, French, Chinese, Japanese) that are preferably stored in the same location as the media asset.

- By DEFAULT, this movie is to be streamed with English subtitles.
- If the **Auto (Recommended)** option is selected during playback of this movie, the languages for subtitles are automatically selected from the list of the autoselected languages (out of English, French, and Chinese).

  For example, the player/device has a localization setting to French then a most appropriate language for the subtitles from your list of autoselected languages will be automatically chosen during playback — French. If this movie is streamed in China, the Chinese subtitles will be picked up, etc.

To illustrate how the enabled **FORCED** option affects your playback, let us use the example shown in the following figure.



Subtitles

☑ Enable Subtitles

Input format
● VTT
○ DFXP

☑ Enable Default Language

File naming
● Prefix Identifier Eg: **MyAssetName_en.vtt**
○ Suffix Identifier Eg: **en_MyAssetName.vtt**

Supported Languages

LANGUAGE CODE	DESCRIPTION	AUTOSELECT	FORCED	DEFAULT	
en	English	☑	□	●	
fr	French	☑	□	○	Remove
zh	Chinese	☑	☑	○	Remove

- There should be either DFXP/WebVTT file per language, hosted in the same storage location as the media asset, if in a different location should be specified with the appropriate query string.
- File should be named as per the naming convention consisting of language code and media identifier, along with underscore "_" separator. Default media identifier will be the same as the media asset name, Media identifier can be overridden in a query string. If the file name follows different naming convention then the complete file name will be specified in a query string.

Example 2: Setting Up Subtitles During Provisioning on Control Center (Using FORCED option)

In this example, even if the **CC** option is **Off** during playback, the **FORCED** subtitles will automatically be displayed — Chinese. The option to choose these Chinese **FORCED** subtitles will not be displayed during playback.

ⓘ **Note:** Some players do not have the **FORCED** option.

Query strings for setting up subtitles

You can use query strings to configure the subtitles for playback:

- If your media assets and accompanying subtitles' files are stored in different locations;
- To override the setup for Subtitles that is done via UI during creation of configuration.

subtitle_file_pattern

The `subtitle_file_pattern` query string can be one of the two following formats:

```
langcode delimiter identifier.extension
```

```
identifier delimiter langcode.extension
```

If subtitle_file_pattern is present

It denotes how the filename of a subtitle segment is constructed:

- *langcode* is a keyword that is replaced with the language codes of that stream.

- *delimiter* is the actual delimiter that is used by the customer.
- *identifier* is a keyword that is replaced either with *subtitle_identifier* or *media identifier*.

> **Note:** All the above 3 fields are required, if the subtitle_file_pattern query parameter is present.

- *extension* is an optional field that would have the actual extension that a customer would like to use. If *extension* is not present, the default extension **vtt** or **dfxp** is used (depending on input format).

If subtitle_file_pattern is absent

The file pattern prescribed in the configuration set up during provisioning on Control Center is used.

subtitle_identifier

The **subtitle_identifier** query string can be only a string.

If subtitle_identifier is present

The *identifier* in **subtitle_file_pattern** is replaced with this value.

If subtitle_identifier is absent

If absent, the media identifier is used to replace the *identifier* in **subtitle_file_pattern**.

subtitle_location

The **subtitle_location** query string can be only a string that present either an absolute or relative directory.

If subtitle_location is present

It denotes the alternate directory location where the subtitle segments are present.

If subtitle_location is absent

The subtitle segments are present in the same location as the media segments.

Query string examples for setting up subtitles

This section provides the examples of query string usage for setting up subtitles.

In all examples presented in the following table:

```
example_url = http://something-vh.akamaihd.net/i/dir1/dir2/
video_,100k,200k,500k,.mp4.csmil/master.m3u8
```

For <*example_url*>, the media files are:

```
cpcode/dir1/dir2/video_100k.mp4
```

```
cpcode/dir1/dir2/video_200k.mp4
```

```
cpcode/dir1/dir2/video_500k.mp4
```

For this example, *media identifier* is `video`. The underscore after `video` is not a part of the media identifier.

The subtitle `delimiter` in these examples is chosen to be an underscore.

The input subtitle format in these examples is assumed to be WebVTT. Hence, the default input subtitle file name extension is `.vtt`.

Query string examples for setting up subtitles

Use case	URL with example query parameters	Subtitle location	Subtitle identifier	Subtitle Filename
`subtitle_file_pattern=langcode_identifier`	*exampleurl*? subtitle_file_pattern=langcode_identifier	Same as media location. *cpcode*/ dir1/dir2/	Same as media identifier `video`	`en_video.vtt` `zh_video.vtt`
`subtitle_file_pattern=identifier_langcode`	*exampleurl*? subtitle_file_pattern=identifier_langcode	Same as media location. *cpcode*/ dir1/dir2/	Same as media identifier `video`	`video_en.vtt` `video_zh.vtt`

`subtitle_file_pattern=identifier_langcode.extension`
`custom file extension = .abc`
	exampleurl?
	subtitle_file_pattern=identifier_langcode.abc
		Same as media location.
		cpcode/ dir1/dir2/
			Same as media identifier `video`	`video_en.abc`
				`video_zh.abc`

If `subtitle_file_pattern` is absent		Same as media location. *cpcode*/ dir1/dir2/	Same as media identifier `video`	File pattern chosen during creating configuration on Control Center.

`subtitle_identifier=subtitles`	*exampleurl*?
	subtitle_file_pattern=identifier_langcode
	subtitle_identifier=subtitles
		Same as media location.
		cpcode/ dir1/dir2/
			subtitle_identifier
			`subtitles`
				`subtitles_en.vtt`

`subtitle_location=dir3`
(Relative directory)
	exampleurl?
	subtitle_file_pattern=identifier_langcode
	subtitle_identifier=subtitles &subtitle_location=dir3
		New relative directory:	*subtitle_identifier*
			`subtitles`
				`subtitles_en.vtt` under *`cpcode/`* `dir1/`

Use case	URL with example query parameters	Subtitle location	Subtitle identifier	Subtitle Filename
		cpcode/dir1/dir2/dir3		dir2/dir3
subtitle_location=/dir3 (Absolute directory)	*exampleurl*?subtitle_file_pattern=identifier_langcode subtitle_identifier=subtitles&subtitle_location=/dir3	New absolute directory: *cpcode*/dir3	*subtitle_identifier* subtitles	subtitles_en.vtt under *cpcode*/dir3

Using closed captions

Closed captions is a process that allows insertion of data/text that corresponds to the video scene. This can be done to provide accessibility to hearing-impaired persons (close captions or open captions), language translations for multi-region programming, or any other reason a producer chooses. Closed captions can be passed to a viewer in one of these ways:

- Embedded into the video file itself, or
- Delivered in an accompanying “sidecar” file

Support for closed captions

Pre-generated “sidecar files” formatted as DXFP can be delivered via PMD alongside the HD video. The player has to do the work to fetch and synchronize the “sidecar” files. There are standard OSMF plugins that handle this functionality. HDCore has APIs support for the captioning/subtitling plugin as well. Refer to the sample under \samples\HDS\AkamaiDFXPCaptioningSample folder in the HDCore package.

Frequently asked questions

How do I allow only the highest bit rates when in full screen mode?

Assume you have a player that has a stage video of 640x360 and an MBR set of files at these rates:

- 640x360, 500 kbps
- 640x360, 1.5mbps
- 1280x720, 2.5mbps
- 1280x720, 3.5mbps

It would be a waste of bandwidth allowing the 3.5mbps feed to play in the 640x360 stage. The end user wouldn't notice a quality benefit and the player would have to spend CPU cycles throwing away most of the video content it was receiving.

1. To limit the player to use the 1280x720 renditions in fullscreen mode, set the **maximumBitrateAllowed** property to **1700** when the HDNetStream class is instantiated. (Any

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

number greater than 1500 will do.) This limits the class's switching algorithms to only allow a bit rate encoded at 1700 or below, and it will only cycle between the two lower renditions.

2. In your player, set a listener on the fullscreen event, and once the player is in fullscreen mode, set **maximumBitrateAllowed** to any number greater than 3500. The class can switch up to the HD resolutions if the user has sufficient bandwidth.

3. Once the user exits fullscreen mode, then you could set **maximumBitrateAllowed** back to 1700. The class switches down to the highest bit rate it can sustain that is lower than that value, effectively limiting the user to a 640x360 video.

How do I determine which index the class uses when it starts a rendition set?

By default, the HDNetStream class starts a rendition set at the highest bit rate, that is less than 500 kbps. If you want to modify this behavior, you have two options:

- Before calling **play()**, set the **startingIndex** property to the index value you want to use. Index values are zero-based, with the renditions ordered by ascending bit rate.

- For a more sophisticated entry point selection, extend HDNetStream and then override the protected**getStartingIndex()** function.

How can I debug what is happening within the HDNetStream class?

The class dispatches an event called **DEBUG**. This contains a data string that traces the internal activity of the class. This is extremely useful information to provide to Media Services On Demand: Stream Packaging when debugging. It is highly recommended you create a mode in your player allowing you to listen to and trace out these debug messages, or perhaps add them to your existing logger.

If I view HD headers, I see many requests being made during playback. Why is that?

Each HDNetStream class will set up two concurrent HD sessions to the edge server. The first, called the delivery session, is the one that delivers the actual video content. This stays connected as long as the video is being downloaded and played. The second session, called the control session, is used by the class to communicate with the server. This communication channel allows it to control the buffer length on the delivery session and also to switch the content that is playing on the delivery session to another bit rate.

What is the timeout value on the Media Services On Demand edge servers?

This is currently two minutes. If you pause the stream for more than 120 seconds, the server times out. If you press play again, the class plays whatever buffer it had built up and then receives a **TIMEOUT** marker from the server. At this point, it automatically reconnects and resumes playback at the point before the timeout.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Set up the video playback for HLS output

Unlike HDS and HD Flash 1.0. playback, playback for HLS streams is accomplished with end users' native media players. So, no direct action on the playback client is required on your part. The following sections will be helpful to you in setting up other aspects of your streams' playback.

Helpful definitions for HLS output

To facilitate your use of this user guide and Media Services On Demand: Stream Packaging itself, this section introduces you to some key terms and definitions you may find useful.

File Type	Description	Extension	Example
Variant Playlist	This file, which the player requests just once at the start of the session, contains a list of all the requested stream's available bit rates (tracks). Since the file includes each track's bandwidth, the player uses this bandwidth information to switch between the most appropriate bit rates in case of network bandwidth fluctuations.	m3u8	master.m3u8
Child Playlist	Each track has its own child playlist file that contains a list of segments available for playback. The number of segments in the file indicates duration of the content.	m3u8	index_1000kpbs.m3u8
Segment	This file contains the video and/or audio data in MPEG2-TS format. For audio-only tracks, the data is contained in AAC format with an .aac extension.	ts / aac	segment10.ts
Encryption Key	This file includes a 128-bit key used to encrypt the segments. It is only requested when encryption is turned **on** at the encoder.	key	crypt.key

Using other compatible playback methods

- HLS Protocol, Version 2 and above

  Refer to *IETF Internet Draft of the HD Live Streaming Protocol Specification*.

- iOS, Version 3 and above

- HTML5

  Supports the *video* tag in the Apple Safari® browser on the Apple OS X® operating system.

- Apple QuickTime® application program on Apple Mac OS® operating system and iOS devices

Using Media Services On Demand support players

You can use the provided support video player to test and troubleshoot your streams: *iDevices Support Player*

Serving your content

Any device that supports HLS output will directly support a segmented streaming approach and launches when you enter an M3U8/M3U link into the Apple Safari® browser. Publishers can also choose to write their own native iPhone applications that are able to seamlessly browse content, leveraging the Media Player for content rendering.

Using SMIL files

This is the usual form for passing your streams' information to the end-user client. For Apple HTTP Streaming, the SMIL publish URL takes the form:

```
http://customer-vh.akamaihd.net/i/smil_file_name.smil/master.m3u8
```

So an actual SMIL URL might be something along the lines of:

```
http://example-vh.akamaihd.net/i/example2a.smil/master.m3u8
```

Following is an example of the format for the contents of SMIL file—example2a.smil:

```
<?xml version="1.0"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN" "http://www.w3.org/2001/
SMIL20/SMIL20.dtd">
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
<head> <meta name="title" content="Event" />
<meta name="httpBase" content="http://example.akamaihd.net/" />
</head>
<body>
<switch id="whatever">
<video src="subdir/example2a_300000.mp4" system-bitrate="300000"/>
<video src="subdir/example2a_500000.mp4" system-bitrate="500000"/>
<video src="subdir/example2a_800000.mp4" system-bitrate="800000"/>
<video src="subdir/example2a_1000000.mp4" system-bitrate="1000000"/>
</switch>
</body>
</smil>
```

Using the client-side URL syntax

If you are unable to generate or provide server-side SMIL files, you can alternatively use a client-side URL format with which you can encode the entire SMIL contents in a single URL. For HLS, the client-side URL takes the form:

```
http://customer-vh.akamaihd.net/i/subdirectory/common_filename_prefix,
bitrate,bitrate,bitrate,bitrate,
common_filename_suffix.csmil/master.m3u8
```

So a published URL to the device for this form of integration might look like:

```
http://example-vh.akamaihd.net/i/movies/example2a_,
300000,500000,800000,1000000,_event1.mp4.csmil/master.m3u8
```

Using the above examples and assuming a CP code of 9389, your content would reside on NetStorage at `example.download.akamai.com/9389/movies/` as:

- `example2a_300000_event1.mp4`
- `example2a_500000_event1.mp4`
- `example2a_800000_event1.mp4`
- `example2a_1000000_event1.mp4`

ⓘ **Note:** If you plan to stream just a single bitrate file, it does not need client-side URL and could be added by itself (that is a single mp4 file called hello.mp4 could have a syntax of *`hostname`*`/i/hello.mp4/master.m3u8`).

HLS delivery improvements per HLS spec upgrade

As part of HLS delivery improvements, a new query parameter `set-akamai-hls-revision` has been introduced to enable HLS manifest improvements and HLS specification upgrades. This query parameter specifies the HLS revision

ⓘ **Note:** `set-akamai-hls-revision` number does not correspond to the HLS Spec Upgrade version number. It is an internal company setting that results in the following:

- If `set-akamai-hls-revision` is set to **4**, the HLS playlist version (EXT-X-VERSION) is **3**;
- If `set-akamai-hls-revision` is set to **5**, the HLS playlist version (EXT-X-VERSION) is **4**.

When you specify the `set-akamai-hls-revision` query string, the following features are automatically enabled as listed in the following table.

Features enabled with revision 4 and 5

Feature	Playlist tag name
If `set-akamai-hls-revision` is set to **4** or **5**, these features are enabled:	
Qualify to enable closed captions control option in stream-inc	#EXT-X-STREAM-INF:<attribute-list>
Remove PROGRAM-ID attribute from EXT-X-STREAM-INF tag	#EXT-X-STREAM-INF:<attribute-list>
Put AVERAGE-BANDWIDTH attribute to EXT-X-STREAM-INF tag	#EXT-X-STREAM-INF:<attribute-list>
If `set-akamai-hls-revision` is set to 5, this additional feature is also enabled:	

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Feature	Playlist tag name
The CODEC parameters that contain the profile and level values for H.264 streams would be advertised in conformance to RFC 6381.	#EXT-X-VERSION:<version> #EXT-X-STREAM-INF:...,CODECS=<codecs>

Note: If a decision is made to advertise independent segments that are key frame aligned, care must be taken to ensure that the source content has been encoded with key frames at intervals that are “factors of the segment duration”. For example, if the chosen segment duration is 10 seconds, then the source content should have been generated with key frames at 1, 2, or 5-second intervals (the GOP size should be 1, 2, or 5 seconds).

The following example shows a playback with the set-akamai-hls-revision query string set to 5.

```
http://example-vh.akamaihd.net/i/xyz.mp4/index_0_a.m3u8?set-akamai-hls-
revision=5
```

Using query strings

There are several query strings available for use with your client-side URL for HLS output:

Note: The values below are in kbps and are applied to the combined audio + video bit rate.

Bandwidth correction (bwcorrection)

This query string can be used to adjust the bitrate advertised in the Variant Playlist’s BANDWIDTH attribute (values are space-delimited). Values can be from 0 to +/-100.

```
uri/master.m3u8?bwcorrection=5 10-10
```

Note: When used, this will be the last filter applied.

Bit rate filtering (b)

If you would like to filter out certain bit rates—to test a particular bit rate version, for example—you can do so by adding a `b` query string to the end of the publish URL. There are four uses:

- Filter all bit rates except those specifically designated (and the lowest audio rate); the string takes the form:

  ```
  b=bitrate,bitrate,bitrate,...
  ```

  So, for example, if you had bit rates 200, 300, 700, and 1200, and you wanted to include only bit rates 200, 700, and the lowest audio rate, your URL would appear as:

  ```
  uri/master.m3u8?b=200,700
  ```

This would produce only the 200 and 700 bit rate streams, as well as the lowest audio bit rate.

- Designate a bit rate range (in kbps) outside of which the stream or streams you wish to filter fall; the string takes the form:

  ```
  b=lower_end-upper_end
  ```

  So, for example, if you had bit rates 200, 300, 700, and 1200, and you wanted to include all bit rates at or greater than 200 and at or less than 700, your URL would appear as:

  ```
  uri/master.m3u8?b=200-700
  ```

  This would produce the 200, 300, and 700 bit rate streams.

- Designate the low end of a bit rate range (in kbps) where all bit rates below the value are filtered; the string takes the form:

  ```
  b=low_end-
  ```

  So, for example, if you had bit rates 200, 300, 700, and 1200, and you wanted to include all bit rates at and higher than 300 kbps, your URL would appear as:

  ```
  uri/master.m3u8?b=300-
  ```

  This would produce the 300, 700, and 1200 bit rate streams.

- Designate the high end of a bit rate range (in kbps) where all bit rates at and above the designation are filtered; the string takes the form:

  ```
  b=0-high_end
  ```

  So, for example, if you had bit rates 200, 300, 700, and 1200, and you wanted to include all bit rates lower than or equal to 700 kbps, your URL would appear as:

  ```
  uri/master.m3u8?b=0-700
  ```

  This would produce the 200, 300, and 700 bit rate streams.

- Any combination of the above.

  So, for example, if you had bit rates 200, 300, 700, 1200, and 1800, and you wanted to include all bit rates except 700, your URL could appear as:

  ```
  uri/master.m3u8?b=0-300,1200-
  ```

  This would produce only the 200, 300, 1200, and 1800 bit rate streams, as well as the lowest audio bit rate.

Note: If Bit Rate Filtering and the Extracted Audio-Only features are simultaneously enabled, bit rate filtering is applied first, then audio-only extraction is applied to the remaining bit rates.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Extracted audio-only stream (__a__)

When enabled in a Media Services One Demand (Stream Packaging) configuration, the Edge server automatically generates an audio-only stream from one of your bit rate streams. You may use the `__a__` query string to disable this feature on a per-stream basis, if desired. Available values are 'on" and "off." For example:

```
uri/master.m3u8?__a__=off
```

Note: This query string is only effective if you are using the extracted audio-only stream. If Bit Rate Filtering and the Extracted Audio-Only features are simultaneously enabled, bit rate filtering is applied first, then audio-only extraction is applied to the remaining bit rates.

Flexible playback (start | end)

This query string enables end-users to begin playback at a specific time (in seconds) within the VoD stream. For example:

```
uri/master.m3u8?start=30&end=500
```

Note: The start time is mandatory but the end time is not. If the end time is not provided, the stream will play until the end of the VoD asset.

Master playlist optional attribute removal (attributes)

Each bitrate in the Master playlist is represented by different attributes, for instance, BANDWIDTH, CODECS, and RESOLUTION. Some of these attributes are optional as per HLS specification. This query string when set to **off** will remove the optional attributes. By default, this option is **on**.

Note: It has been observed that certain devices and platform, like XBOX 360, do not ignore these optional tags if the implementation does not support them. This behavior is not specification complaint. Use of attributes=off query string would let you play the content on these devices by removing the optional attributes in this file.

For example, applying this string query to the following attributes highlighted in bold:

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-
ID=1,BANDWIDTH=1128000,RESOLUTION=960x540,CODECS="avc1.77.30, mp4a.40.2"
http://example-vh.akamaihd.net/i/movies/event.smil/index_1000_av-p.m3u8?sd=10
#EXT-X-STREAM-INF:PROGRAM-
ID=1,BANDWIDTH=2528000,RESOLUTION=1280x720,CODECS="avc1.77.30, mp4a.40.2"
http://example-vh.akamaihd.net/i/movies/event.smil/index_2400_av-p.m3u8?sd=10
#EXT-X-STREAM-INF:PROGRAM-
ID=1,BANDWIDTH=3128000,RESOLUTION=1280x720,CODECS="avc1.77.30, mp4a.40.2"
http://example-vh.akamaihd.net/i/movies/event.smil/index_3000_av-p.m3u8?sd=10
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=139000,CODECS="mp4a.40.2"
http://example-vh.akamaihd.net/i/movies/event.smil/index_1000_a-p.m3u8?sd=10
```

will result in the following:

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1128000
http://example-vh.akamaihd.net/i/movies/event.smil/index_1000_av-p.m3u8?sd=10
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2528000
http://example-vh.akamaihd.net/i/movies/event.smil/index_2400_av-p.m3u8?sd=10
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=3128000
http://example-vh.akamaihd.net/i/movies/event.smil/index_3000_av-p.m3u8?sd=10
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=139000
http://example-vh.akamaihd.net/i/movies/event.smil/index_1000_a-p.m3u8?sd=10
```

Preferred bit rate (__b__)

If desired, you can designate a specific bit rate stream to be the “preferred” bit rate, meaning the end user’s player would start with this bit rate before adjusting to another bit rate depending on network conditions. For example:

```
uri/master.m3u8?__b__=500
```

Note: This query string is ineffective with Microsoft® Xbox® video player, as it automatically selects the lowest bit rate at the beginning.

Testing your content

You should test your content prior to making it available to your end-users to help ensure they will have a problem-free experience.

Note: Staging hostnames are added automatically for all **akamaihd.net** domains in order to test configurations on the Staging network. For example, if your akamaihd hostname is **example-vh.akamaihd.net**, your staging hostname will be example-vh.akamaihd-staging.net.

How to

1. Compose your URLs and request the .m3u8 and segment files directly from a Web browser to check if they are cached and of the correct duration. This will confirm whether the connection delivery is correct. Be aware, you will not see video playback during this test.
2. Using the iPhone emulator, iPhone, iPad, or iPod Touch® with the latest iOS version, request the file to watch it and ensure it plays correctly.

Configure security

Security for HLS output is provided by the SecureHD Policy Editor (SPE), which is configurable through Control Center. For information regarding SPE and its configuration, refer to *Media Security Policy Editor User Guide* also available in Control Center.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

ⓘ **Note:**

- SPE's Player Verification does not currently support HLS output. Also, Encryption Percentage, a part of SPE's Media Encryption feature, does not apply to Apple HTTP Streaming.
- SecureHD Media Encryption feature is not supported for HLS with Additional Digital Properties, meaning, the hostname does not end with akamaihd.net or akamaihd-staging.net domain
- It is recommended that you do not have more than 10 hostnames associated with SecureHD Policy Editor for performance reasons. If you require more than 10 hostnames associated with SecureHD Policy Editor, contact your Account Representative for assistance.

Using closed captions

Closed captions is a process that allows insertion of data/text that corresponds to the video scene. This can be done to provide accessibility to hearing-impaired persons (close captions or open captions), language translations for multi-region programming, or any other reason a producer chooses. Closed captions can be passed to a viewer in one of these ways:

- Embedded into the video file itself, or
- Delivered in an accompanying “sidecar” file

Support for closed captions

In the h.264 input stream, if the subtitles text is present in the NALU header type 6 (SEI) as ATSC picture user data, it will be passed through and available in the HLS output.

Purging HDS and HD Flash 1.0 video content

Note: The best practice is always to upload the updated content under a new name and then update your URL accordingly.

However, there might be situations in which you will want to replace new content while using the same file name. This would involve uploading the new content in place of the old content.

However, edge servers might retain the old content in their caches, requiring you to purge those caches to allow end-users to play the new content.

If you do decide to use in-place content purge, the next section describes the procedures. Also, note that in-place cache purging can be time intensive (approximately 90 minutes).

Note: Because of the nature of the multiple bitrate feature, you must purge every version of the content in question. Failure to do so can result in unpredictable behavior.

There are different procedures for purging content from the streaming cache depending on what formats you are using: HDS or HD Flash 1.0.

Purging streaming cache of HDS content

To illustrate the procedures below, the following assumptions are used as an example.

The manifest URL:

```
http://example-vh.akamaihd.net/z/example/event1.smil/manifest.f4m
```

And these files residing on the origin:

- event1_220K_video.mp4
- event1_500K_video.mp4
- event1.smil

Note: Be certain you use the provided hostname and not your own CNAME'd hostname in the purge URL.

Use Control Center's HD Content Control Utility to purge your Flash media.

How to

1. Replace the content in question on your origin.
2. If you are using NetStorage for your origin, refer to *NetStorage User Guide* for information regarding accessing your account.

3. Log in to Control Center.
4. In the upper navigation bar, click the **PUBLISH** tab. The **Publish** menu appears.
5. Select **Content Control Utility**. The **Select Product** page appears.
6. Select the **HD On Demand Streaming** radio button and click **Continue**. The **HD Content Control Utility** page appears.
7. Select the **HD On Demand Streaming** radio button and click **Continue**. The **HD Content Control Utility** page appears.
8. From the **Content Type** drop-down menu, select **Adobe Dynamic Streaming**.
9. If desired, enter a name for the request in the **Request Name (Optional)** text box.
10. In the **Content to Refresh** area:

If...	Then...
You want to purge by URL/ARL	1. Select the URL/ARL entered below radio button. 2. In the first text box, enter the manifest URL from which content is to be purged. For example: `http://example-vh.akamaihd.net/z/examplehost/event1.smil/manifest.f4m` 3. In the second text box, enter the file names to be purged, separating each with a newline. For example: • SMIL: event1_220K_video.mp4, event1_500K_video.mp4, event1.smil • Single File: event1_220K_video.mp4 • Client-Side URL Syntax: example2a_,300000,500000,800000, 1000000,_event1.mp4.csmil • Manifest: Specify the manifest (.f4m), index (.f4x), and data files (.f4f) – Manifest: For example, event1.f4m – Index: For example, event1_220K_videoSeg1.f4x – Data: For example, event1_220K_videoSeg1.f4f
You want to purge all URLs/ ARLs.associated with specific CP codes	1. Select the Refresh ALL URLs/ARLs Associated with Specific CP Codes radio button. 2. Select the check box of any CP codes for which you would like to purge content.

If...	Then...
ⓘ **Note:** Use care when using this feature.	

11. In the **Notification** area, select whether you would like to receive an e-mail notification when the content purge is complete.

    If you choose to receive notification, enter your e-mail address in the accompanying text box.

12. Click **Start Refreshing Content**.

What you should see

The purge is initiated. An e-mail will be forthcoming if you opted to for email notification to signal the completion of the process.

ⓘ **Note:** Be aware that cache purges can take approximately 90 minutes to take effect.

Purging the streaming cache of HD Flash 1.0 content

Before you begin

Before purging the streaming cache of HD Flash 1.0, you must construct a purge URL for your content.

The purge URL is fairly simple to construct, simply being the hostname, path, and file name of the content you wish to remove in the form:

```
http://Akamai_hostname/directory/directory/.../content_name.
```

ⓘ **Note:** Be certain you use the provided hostname and not your own CNAME’d hostname in the purge URL.

An example of a purge URL might be:

```
http://examplehost-vh.akamaihd.net/movies/trailers/trailer1.mp4
```

This is the URL you will use in the HD CCU.

How to

1. Access the HD Content Control Utility:

    a. Log in to Control Center.

    b. In the upper navigation bar, click the **PUBLISH** tab.
    The **Publish** menu appears.

    c. Select **Content Control Utility**.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

The **Select Product** page appears

d. Select either the **Media Services On Demand** radio button and click **Continue**.
The **HD Content Control Utility** page appears.

2. Purge the Flash video from the streaming cache.

a. From the **Content Type** drop down menu, select one of the following:

- HD Flash 1.0
- HLS
- HDS

b. In the **Content to Refresh** area, enter your purge URL in the box; separate multiple purge URLs with a new line.

c. In the **Notification** area, select the desired radio button to either receive or not receive an e-mail notification upon removal of the content. If you choose to receive notification, enter your e-mail address in the text box.

d. Scroll to the bottom of the page and click **Start Refreshing Content**.The content will be purged from Media Services On Demand infrastructure. Afterward, the next end-user request for the video will cause the service to download the new content from your Origin location.

ⓘ **Note:** Be aware that cache purges can take half an hour or longer to take effect.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Purging HLS video content

ⓘ **Note:** The best practice is always to upload the updated content under a new name and then update your URL accordingly.

However, there might be situations in which you will want to replace new content while using the same file name. This would involve uploading the new content in place of the old content.

However, Edge servers might retain the old content in their caches, requiring you to purge those caches to allow end-users to play the new content.

If you do decide to use in-place content purge, the next section describes the procedures. Also, note that in-place cache purging can be time intensive (approximately 90 minutes).

ⓘ **Note:** Because of the nature of the multiple bitrate feature, you must purge every version of the content in question. Failure to do so can result in unpredictable behavior.

To illustrate the procedures below, the following assumptions are used as an example.

The manifest URL:

```
http://example-vh.akamaihd.net/i/examplehost/event1.smil/master.m3u8
```

These files are residing on your origin:

- event1_220K_video.mp4
- event1_500K_video.mp4
- event1.smil

ⓘ **Note:** Be certain you use the Media Services On Demand: Stream Packaging hostname and not your own CNAME'd hostname in the purge URL.

Use Control Center's HD Content Control Utility to purge your media.

How to

1. Replace the content in question on your origin.
2. If you are using NetStorage for your origin, refer to *NetStorage User Guide* for information regarding accessing your account.
3. Log in to Control Center.
4. In the upper navigation bar, click the **PUBLISH** tab. The **Publish** menu appears.
5. Select **Content Control Utility**. The **Select Product** page appears.
6. Select the **HD On Demand Streaming** radio button and click **Continue**. The **HD Content Control Utility** page appears.

7. From the **Content Type** drop down menu, select **HD Apple iPhone/iPad On Demand**.
8. If desired, enter a name for the request in the **Request Name (Optional)** text box.
9. In the **Content to Refresh** area:

If...	Then...
You want to purge by URL/ARL	1. Select the **URL/ARL entered below** radio button. 2. In the first text box, enter the manifest URL from which content is to be purged, for example: `http://example-vh.akamaihd.net/examplehost/event1.smil/master.m3u8` 3. In the second text box, enter the file names to be purged, separating each with a newline. For example: • SMIL: event1_220K_video.mp4, event1_500K_video.mp4, event1.smil • Single File: event1_220K_video.mp4 • Client-Side URL Syntax: example2a_,300000,500000,800000,1000000,_event1.mp4.csmil
If you want to purge all URLs/ARLs associated with specific CP codes **Note:** Use care when using this feature.	1. Select the **Refresh ALL URLs/ARLs Associated with Specific CP Codes** radio button. 2. Select the check box of any CP codes for which you would like to purge content.

10. In the **Notification** area, select whether or not you would like to receive an e-mail notification when the content purge is complete.

    If you choose to receive notification, enter your e-mail address in the accompanying text box.
11. Click **Start Refreshing Content**.

    The purge is initiated. An e-mail will be forthcoming if you opted to for e-mail notification to signal the completion of the process.

    Note: Be aware that cache purges can take approximately 90 minutes to take effect.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Reporting

Several historical data reports are available to you via Akamai Control Center.

These include traffic reports, visitor reports, URL reports, and URL locations reports. You view various data about your MSOD instance and you can customize your reports. Usage for *HDS/HLS* and *HD Flash 1.0* content reports are covered in the *Media Services Reports User Guide*.

Source: https://techdocs.akamai.com/msod/pdfs/msod-stream-packaging-user-guide.pdf

Notice

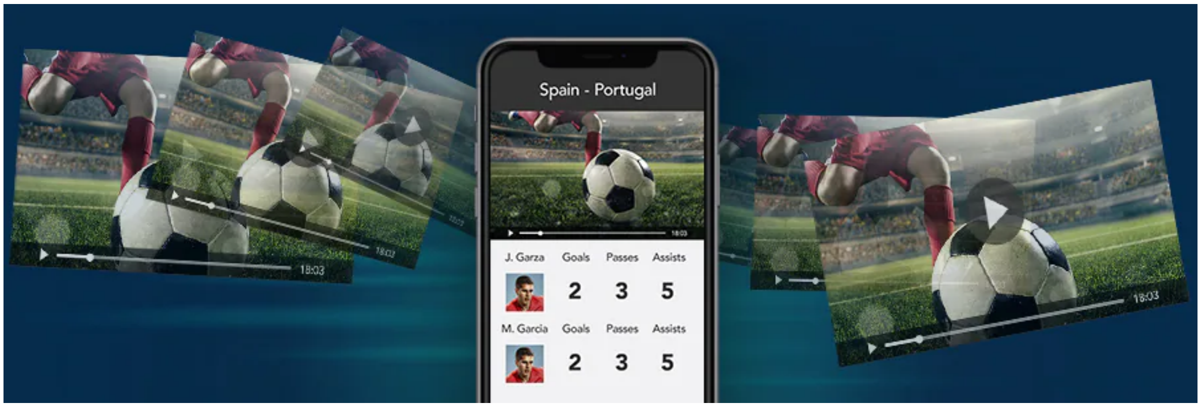
Akamai secures and delivers digital experiences for the world's largest companies. Akamai's Intelligent Edge Platform surrounds everything, from the enterprise to the cloud, so customers and their businesses can be fast, smart, and secure. Top brands globally rely on Akamai to help them realize competitive advantage through agile solutions that extend the power of their multi-cloud architectures. Akamai keeps decisions, apps, and experiences closer to users than anyone — and attacks and threats far away. Akamai's portfolio of edge security, web and mobile performance, enterprise access, and video delivery solutions is supported by unmatched customer service, analytics, and 24/7/365 monitoring. To learn why the world's top brands trust Akamai, visit *www.akamai.com*, *blogs.akamai.com*, or *@Akamai* on Twitter. You can find our global contact information at *www.akamai.com/locations*.

Akamai is headquartered in Cambridge, Massachusetts in the United States with operations in more than 57 offices around the world. Our services and renowned customer care are designed to enable businesses to provide an unparalleled Internet experience for their customers worldwide. Addresses, phone numbers, and contact information for all locations are listed on *www.akamai.com/locations*.

Published 1/2022

Exhibit R

Media Delivery



Adaptive Media Delivery

Adaptive Media Delivery is an HTTP-based video streaming solution optimized for live and VOD Adaptive Bitrate (ABR) streaming to provide high-quality viewing experiences across a variety of networks and device types. Adaptive Media Delivery offers a 100% availability SLA, IPv6 support, token authentication, access control, site failover, and much more in four formats: HTTP live streaming, HTTP Dynamic Streaming (HDS), Microsoft Smooth Streaming, and MPEG-DASH. Test drive Adaptive Media Delivery with our free trial
.

DOCUMENTATION Product Overview

Download Delivery

Download Delivery delivers large (>100 MB) file-based content over the Internet. It is built on the globally distributed Akamai intelligent edge platform for superior capacity, scalability, availability, and performance. It provides clear, comprehensive metrics and optional tools that can monitor and manage the entire download process across your customer base. Download Delivery ensures a predictable, high-quality download experience while helping you address your online distribution goals.

DOCUMENTATION Product Overview

NetStorage

Akamai's NetStorage solution provides petabytes of storage space in Akamai data centers, automatically replicating the data all over the world to provide the fastest, most reliable origin-transfer times possible. An essential element to scaling Akamai's media delivery solutions is an origin storage solution that is optimized for fast, reliable transfers. NetStorage is designed to operate in concert with Akamai's delivery capabilities to specifically provide the highest performance and uptime possible for your online delivery applications.

DOCUMENTATION Product Overview

Media Services Live

Akamai's Media Services Live lets your users view near-real-time audio and video content with media players such as Adobe Flash, Microsoft Silverlight, and Apple iPhone technologies. It's specifically designed to meet the challenges of 24/7 live linear broadcasts online, delivering secure and reliable transport of content with minimal delays.

DOCUMENTATION Product Overview

Adaptive Media Player

The Akamai Adaptive Media Player (AMP) is designed to empower content providers by simplifying their player operations and ensuring quality online media playback for web or app-based streaming experiences. In addition to optimizing the bitrate playback algorithms, AMP's integrated stream protection, analytics, integration, and closed-caption support allow easy adoption of additional online media services without the headaches of staying on top of the shifts in this rapidly changing technology landscape. Take Adaptive Media Player for a test drive in our free trial.

DOCUMENTATION Product Overview

Sign up for the Akamai Developer Program today.

REGISTER

COMPANY

Akamai.com

Akamai Locations

Contact Us

DEVELOPER

Documentation

Blog

Release Notes

STAY IN TOUCH

SELECT LANGUAGE ▼

Exhibit S

(Reserved)

Exhibit T

Akamai DEVELOPER

Optimization Overview

Learn

The following is a non-exhaustive summary list of optimizations on the Front End Optimization (FEO) platform.

File versioning

All resources that are optimized are renamed with a unique name. If the object changes, a different uniquely named resource will be created. These files have great cacheability, typically set with 365days for time to live (TTL).

Sharding

Sharding is the practice of serving resources from multiple domains to circumvent browser connection limitations.

- For example, www.example.com's resources are served from 1.example.com and 2.example.com, increasing from six connections to eighteen connections in some browsers.

Cookieless Shard Domains

Cookieless sharding is a practice of serving resources from a domain where cookies will not be sent from the browser when requesting a resource

- For example, www.example.com's resources might be served from www.exampleimages.com
- The primary benefit of using cookieless domains is to reduce the number of bytes sent and eliminate the time to process the cookies, which are generally not needed for static resources.
- In practice, these are often off-domain to eliminate the risk of a wildcard path on third-party cookies.

DNS Prefetch

DNS Prefetch references can be injected at the top of the document allowing the browser to do early lookups. DNS Prefetch content is automatically detected by the FEO analyzer.

JavaScript Optimizations

The Front End Optimization platform contains a wide variety of optimizations that improve JavaScript resources using a variety of techniques, such as

- Minification - Removing whitespace and comments to reduce file size
- HTML5 - Improving caching
- Adaptive Consolidation - Reducing roundtrips to retrieve all the scripts by consolidating scripts.
- Asynchronously load JavaScripts to speed rendering of a page
- Pre-Execution - Execute scripts server side to improve client side performance
- Resource prefetching - Speed up site navigation by preloading JS/CSS resources for other pages.

CSS Optimizations

In addition to the Minification, HTML5 features, Adaptive Consolidation and CSS optimizations, the platform can

- Asynchronously load the CSS files to improve rendering
- Small image embedding to reduce requests
- Defer print style sheets to render display critical resources first

Image Optimizations

Front End Optimization can analyze images and provide a number of improvements

- Select the optimal image format (webp, JPEG XR, JP 2000)
- Reduce resolution
- Responsive images - sizing images for the appropriate screen size
- On-demand images - only loading images when they come into view

EdgeStart

The EdgeStart optimization allows Akamai to use a combination of caching the initial static part of a non-cacheable page with streaming the remainder of the response from origin to greatly improve rendering and time to first byte for non-cacheable pages.

COMPANY

DEVELOPER

Akamai.com
Akamai Locations
Akamai Privacy Statement
Contact Us

Documentation
Blog
Release Notes

STAY IN TOUCH

SELECT LANGUAGE ▼

Exhibit U

AKAMAI WHITE PAPER

Front-End Optimization on the Akamai Intelligent Platform™

Akamai
FASTER FORWARD

TABLE OF CONTENTS

Introduction

In the hyperconnected world, web and mobile application performance is playing a more critical role than ever in driving user adoption and engagement. Today's savvy audiences demand richer, more interactive experiences that require more data and more processing. This is a trend of ever-increasing average web sizes. Yet consumers continue to expect pages to load faster than ever before, indiscriminately abandoning sluggish sites. At the same time, studies have repeatedly shown that enhancing site performance brings about greater page views, better audience engagement, higher conversion rates, and even improved SEO rankings.[2] Indeed, as little as a few tenths of a second can make a difference.[3]

Unfortunately, understanding website performance is no small task, particularly as the Internet and the applications running across it become more and more complex. Today, a surprisingly large number of factors – including network latencies, communications protocols, device capabilities and browser page rendering behaviors – play a role in determining the end user perception of website performance.

Traditionally, web performance tuning efforts focused on the application back end. This is the area over which web developers have the greatest control, but unfortunately, changes made here may not translate into big improvements for the end user. This is because back-end time – which includes the overhead involved in making database calls and generating the HTML page – actually accounts for only about 10% of user-perceived page load time for the typical web application today. The remaining 90% is due to middle mile and front end latencies. The middle mile refers to the time it takes for data to travel across the Internet, which can face significant delays due to network latency, routing problems, and Internet congestion. The front end includes the time it takes for data to cross the last mile to the user's device and then for the browser to render the page. website performance optimization needs to successfully address the 90% problem in order to deliver the level of responsiveness end users demand.

How website page load time breaks down

Back End or First Mile	Middle Mile	Front End or Last Mile
• Database calls • HTML page generation	• Retrieving page content, (including HTML, images, JavaScript, etc.,) from origin server, across the Internet (the 'Long-hair' across multiple networks and peering points)	• Delivering content over cable modem, cellular network, etc., to end user • Rendering page in the browser

What Is Front End Optimization?

As content and application delivery networks have proven effective at helping sites overcome middle mile performance issues, front end performance issues are now coming into focus and being addressed. Front end bottlenecks play a growing role in overall site performance as today's rich and complex Internet applications make increasing use of embedded objects, sophisticated stylesheets, client side code, and third-party content – all of which slow down the rendering of a page. Mobile sites, with their sluggish networks and weaker device processors, face an even greater front end challenge. The need to address these types of performance issues has led to increased awareness about Front End Optimization (FEO), a set of best practices designed to reduce front end delays in order to improve the end user experience.

FEO techniques can be broadly grouped into four complementary approaches:

1. **Reduce the number of HTTP requests.** Before rendering a web page, browsers first make an HTTP request for the HTML page, followed by requests for additional resources required to display the page – such as images, CSS, or JavaScript. The number of requests required for a typical page has more than tripled in the last 9 years, to a current average of 100 objects per page.[5]

   Unfortunately, each new HTTP request can increase unwanted delay in setting up the connection. For mobile devices, this overhead typically takes more time than downloading the page data itself. Even for desktop browsers, this overhead is one of the key factors in determining page rendering times, and reducing the number of requests can greatly improve the user's perception of site responsiveness. FEO employs various techniques to achieve this request reduction, including consolidating multiple files into a single request, caching objects locally using HTML5's Application Cache interface, and inlining small objects within the HTML.

2. **Reduce the size of the data.** Page weight is often the single largest factor in page download times, so reducing the number of bytes transferred can significantly speed up a site. FEO techniques are designed to reduce page weight without changing visual quality or site functionality.

   Images are typically the biggest component of page weight and can often benefit from compression or resizing without compromising quality. Adaptive image compression, where resizing is optimized based on the end user's screen resolution and connection speed, is especially effective. Text blocks – such as HTML, JavaScript and CSS files – can also be significantly reduced in size through techniques such as compression and minification, the process of removing unnecessary characters such as whitespace and comments.

3. **Accelerate page rendering.** Different browsers exhibit different page rendering behaviors, many of which prevent pages from rendering as quickly as possible. For example, browsers often:

   - Block other downloads while downloading, parsing, and executing scripts.
   - Limit the number of simultaneous requests for images and other embedded content. Older versions of Internet Explorer and Firefox, for example, are limited to two parallel requests per hostname. Newer versions increase this limit to 6.
   - Block page rendering while stylesheets are downloaded and parsed.

   FEO techniques address these problems by recoding the HTML in a way that distributes the browser's work so that the page can begin to render as soon as possible, delaying nonessential tasks until later. Techniques include using asynchronous JavaScript execution, which helps prevent scripts from delaying object downloads and page rendering; just-in-time or on-demand image loading, which loads only images in the current viewable area, deferring others until the user scrolls; and domain sharding, or retrieving objects from multiple subdomains to work around browser limitations on concurrent requests. While these techniques work well to minimize page rendering bottlenecks, they do need to be customized to each browser's specific behaviors. In some cases, we may add code to take advantage of newer browser capabilities, while, in other cases, we may make changes to overcome deficiencies in older browsers.

4. **Alleviate third-party bottlenecks.** Ads, analytics code, social media widgets, and other third-party content now represent a large percentage of total page requests. Unfortunately, due to the page rendering behaviors discussed above, sluggish third-party content can significantly slow down a page or even make it appear completely unavailable to the end user, as the browser may block the progress of a page while waiting for the third-party content to load.

   websites have no control over the performance of third-party content, but FEO best practices can alleviate these bottlenecks by decoupling the third-party script execution from the loading of the page. By making the third-party request asynchronous, FEO ensures that a slow third-party tag will not negatively impact the loading of the other objects and the rendering of the page.

   For an overview of Akamai FEO optimizations please refer to the appendix.

FEO Implementation And Challenges

FEO evolves constantly as browser and device technology continues to change, but it is hardly a black box or secret art. Indeed, many FEO best practices are well documented across the web on sites like YSlow, PageSpeed, and Steve Souders' High Performance web Sites blog. In theory, this means any website development team can simply implement the techniques and enjoy significant performance improvements across its site. The reality, however, can be quite different.

First, FEO is complex, making it difficult to implement and costly to maintain. The up-front investment required to manually retrofit an existing site for FEO can be on the order of many man-years – not including the ongoing resources needed to keep the site optimized. Moreover, unless all members of the development team gain FEO expertise, any implemented optimizations may be lost each time the site itself is changed or redesigned.

Second, FEO best practices are not one-size-fits-all. Different approaches are relevant in different situations, and if used incorrectly, FEO techniques can actually hurt performance. For example, domain sharding may work well for desktop browsers running on fast connections, but it may hurt performance in certain mobile scenarios. The ideal number of domains to use also depends on multiple factors.

Generally, FEO techniques involve balancing tradeoffs among many different considerations. For instance, one might assume that inlining all content – that is, requesting all page resources in a single download – is ideal for performance, because it reduces HTTP requests down to one. In reality, this is not optimal for most situations, as you lose the often-greater performance benefits of being able to cache images or use just-in-time image loading to reduce page download size and accelerate page rendering. The balancing point between these tradeoffs can be a complex and moving target, as browser, devices and networks evolve – and proliferate – at a rapid pace. Trying to keep site optimizations up to date in the face of never-ending changes, both in the device/browser landscape and within the application itself – is a daunting and expensive proposition.

Automated Front End Optimization

In an attempt to address some of the challenges associated with manual FEO implementation, a crop of young companies are now providing automated FEO technologies, available either as an appliance to be deployed in front of a site's servers, or as a service that intercepts and responds to requests on behalf of the site. These FEO technologies attempt to tackle only the front end piece of the overall performance puzzle; they do not address network latency, congestion, and reliability issues that can severely hamper the delivery of any website or application, for example. However, they can work in conjunction with content and application delivery networks to provide a more comprehensive solution.

Unfortunately, both appliance and service-based FEO solutions often become bottlenecks of their own. Automated FEO solutions sit in the line of fire, taking the HTML code generated by a site and modifying it before delivering it to the end user. Being in the critical path, the reliability, scalability and performance of the FEO solution is paramount. For service-based solutions, the website is at the mercy of its service provider. For appliance-based solutions, the website must architect and plan for capacity and failover as well as maintain their new boxes. Either way, there is a cost and risk associated.

In addition, these FEO solutions often have integration or compatibility issues with the other components of the website's overall performance strategy, such as the content or application delivery network. When cobbling together two disparate solutions, companies are left to fend for themselves if something goes wrong, troubleshooting in the no-man's land in between the cracks. These situations can create support nightmares for the site, particularly with a technology that modifies website code on the fly. Thus, while stand-alone automated FEO solutions have the potential to provide some performance benefits without the high cost of manual FEO coding, they also carry a number of risks that make them far from ideal.

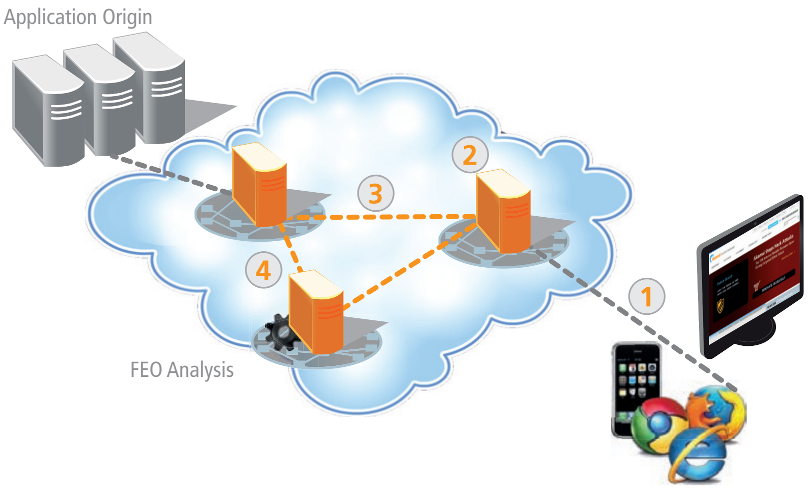
Front End Optimization With Akamai

Akamai takes an end-to-end approach to solving website performance, offering a cohesive, multi-layered performance solution designed to help businesses overcome origin, middle mile, and front end bottlenecks automatically and effortlessly. For every end user request, Akamai's proven technologies are dynamically applied in a way that optimizes performance for that unique scenario, taking into account real-time website, network, and end user conditions. Akamai's FEO capabilities are an integrated part of these solutions, working in concert with our other performance, security, and availability offerings to deliver the best possible experience for every user, on every device, every time.

Akamai's FEO capabilities are based on advances pioneered by Blaze, an industry-leading FEO company that Akamai acquired in early 2012. Blaze's cutting-edge FEO solution was designed from the ground up to work in synergy with a distributed content and application delivery network, providing an ideal architecture for incorporation into the Akamai Intelligent Platform. Built on this foundation, Akamai is now able to offer a uniquely robust and scalable automated FEO solution as a seamlessly integrated part of our comprehensive web performance solutions.

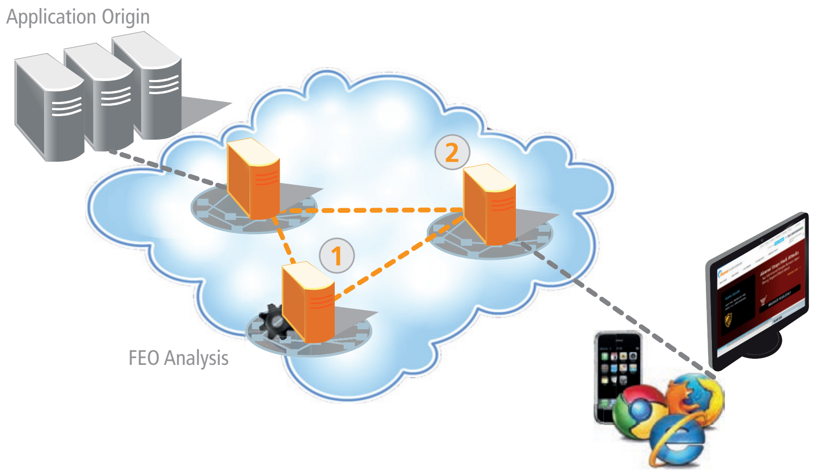
Overview: How it Works

In the diagrams below, we illustrate how Akamai's FEO works:



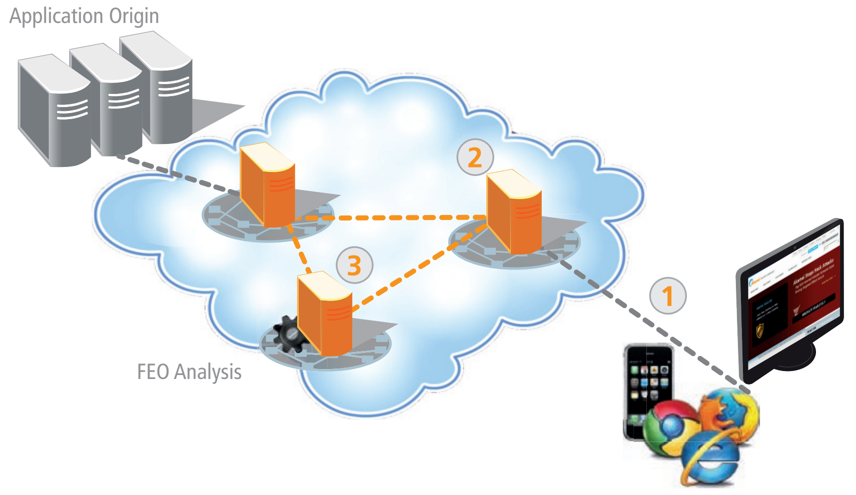
Part I. First Request

1. User requests a page
2. No front end optimizations are available
3. Page is accelerated across the middle mile and delivered to first-request user without front end optimizations
4. Subsequently, page is queued with the FEO Analysis Engine to be analyzed



Part II. Offline Analysis

1. FEO Analysis Engine evaluates page to determine the best ways to optimize it, and produces optimized transformation rules
2. FEO Analysis Engine makes optimized transformation rules available across the Akamai Intelligent Platform



Part III. Subsequent Requests and Continued Analysis

1. Another user requests the page
2. Akamai applies the pre-created front end optimizations and delivers accelerated page with front end optimizations applied
3. Page is continuously monitored and re-analyzed

As always, end user requests are efficiently routed to a nearby Akamai server. This server may be able to deliver the requested content straight from cache or will fetch it from the origin server using route, protocol, and application optimizations to accelerate delivery over the long-haul Internet. The server also applies the relevant front end optimizations before delivering the content to the end user, which enables speedy page rendering. The end result is an exceptional end user experience that is rich, reliable, and responsive, regardless of device or Internet conditions.

Asynchronous Analysis Delivers Speed

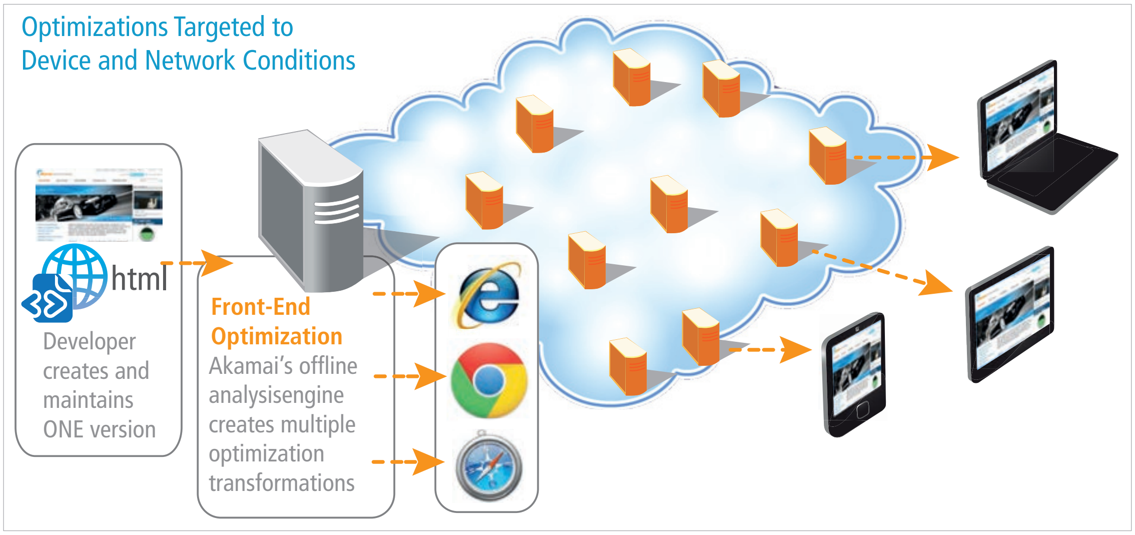
Understanding which front end optimizations to apply to a dynamic site requires complex and computationally intensive analysis. Our sophisticated, offline analysis engine does this work asynchronously, precomputing transformations to be applied to the various static and dynamically generated pages of a site. This information is then provided to servers across the Akamai Intelligent Platform, allowing them to rapidly apply the transformations in real time as they deliver content to end users. By separating the complex analysis from the real-time application of the optimizations, Akamai delivers cutting-edge FEO capabilities without sacrificing speed.

Adaptively Optimized for Each End-User

To be effective, FEO optimizations need to be customized for the end user environment, because different browsers and devices behave differently. For example, browsers differ in their support for features like asynchronous JavaScript, HTTP pipelining or the number of simultaneous downloads they can make, suggesting different optimization behaviors. Images may also be resized differently for different devices and screen resolutions. In addition, a device connected over a loss-y, high-latency 3G network may benefit from different types of optimizations than a device running over a fast Internet connection.

Akamai's platform handles these different situations seamlessly, creating multiple possible optimizations in the analysis phase, and then intelligently applying the appropriate ones to each request based on real-time factors such as the end user's browser version, device and network speed. This means we take advantage of the full capabilities of each user's specific setup to deliver the best possible experience – rather than simply settling for the lowest common denominator supported across all browsers – or targeting only the most popular browsers while leaving other users out in the cold.

Moreover, Akamai continually updates and improves its optimization engine to reflect the state-of-the-art in FEO innovations as well as the constantly evolving browser and device landscape. This enables our customers' development teams to focus simply on creating innovative content and features, rather than getting bogged down by the ever-changing details around which versions of which browsers support what features. Your team creates and maintains just one version of your site; Akamai enables it to work optimally for every single user across an increasingly heterogeneous end user environment.



Optimizations Targeted to Device and Network Conditions

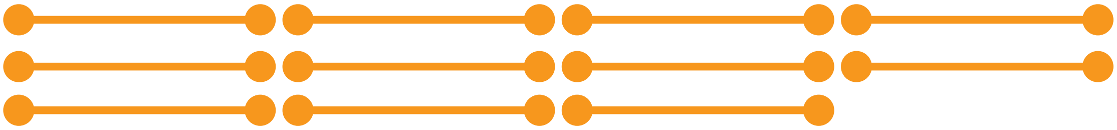
html
Developer creates and maintains ONE version
Front-End Optimization
Akamai's offline analysisengine creates multiple optimization transformations

Overcoming Bottlenecks From End to End

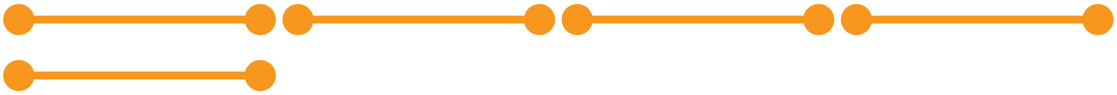
By integrating FEO capabilities directly into the Akamai Intelligent Platform, these front end optimizations work seamlessly with Akamai's entire suite of offerings, providing complementary performance benefits that overcome bottlenecks from end to end on a platform that is highly secure and proven to help deliver reliable, consistent web experiences to end users. FEO has particularly strong synergies with Akamai's route protocol and connection optimizations. These capabilities are designed to deliver every byte faster while Akamai FEO works at the same time to reduce the amount of bytes needed. This holistic, multi-layered approach to website performance is designed to help businesses deliver exceptional end user experiences with unmatched performance, reliability, and scale.

End-to-end Performance Benefits with Akamai

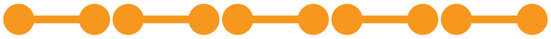
Without Akamai: Many requests, many bytes, many long round-trips



With FEO : Fewer requests, fewer bytes



With FEO + Dynamic Site Accelerator (DSA: Fewer requests, fewer bytes, faster round-trips



Mobile Performance Optimizations

Because of the more restrictive capabilities of its devices and networks, the fast-growing mobile web faces a number of unique performance challenges in the last mile, shown in the table below. Yet consumer expectations remain high: a recent study showed that 71% of mobile users feel sites should load as quickly on their mobile devices as on their desktops. [6]

Front-End/Last Mile Bottlenecks For Mobile

Cellular Network Limitations	Device Limitations	Browser Limitations
• High latency • High packet loss • Short-lived communications channel and high overhead for establishing	• Weak CPUs mean longer script parse times and slower render-ing • Small device caches • Smaller screen sizes and far greater variation in screen sizes	• Greater diversity of browsers and OS versions to support • Browsers support fewer features

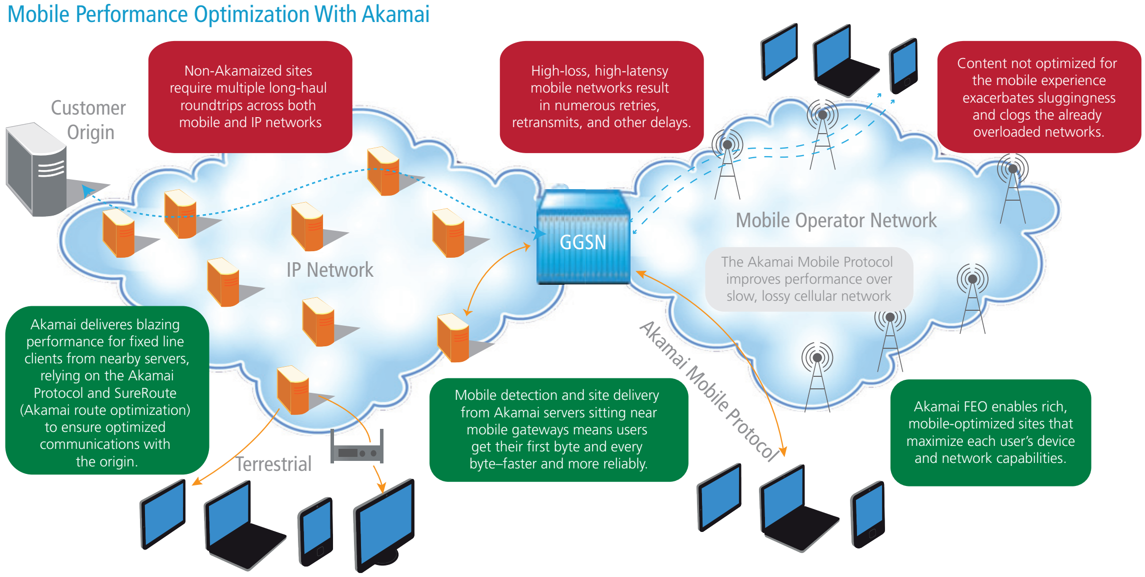
In this type of environment, FEO can have an enormous benefit. Useful techniques include many of the optimizations we have already mentioned – such as adaptively reducing image sizes based on display size or network speed, maximizing use of local device caches, delaying script parsing, and consolidating resource requests – tuned with mobile-specific parameters, as well as additional, mobile-specific optimizations such as:

- JavaScript Pre-Execution: delivering mostly static pages, with script pre-executed on the server wherever possible
- Click On Touch: Converting hyperlinks to click events to eliminate the 300ms delay associated with standard mobile hyperlink clicks
- Cellular Keep-Alive: Keeping mobile connections open longer in between page requests in order to avoid the high overhead of re-establishing a connection

In mobile, it is particularly critical that optimizations are customized for each request. Because the Akamai Intelligent Platform knows about the end user's device, browser, and real-time network situation, it can apply FEO transformations that can help content providers maximize that user's experience based on his or her current environment.

Equally important, these FEO capabilities work seamlessly with Akamai's other mobile technologies, including the Akamai Mobile Protocol, which improves communication speeds over the slow, lossy cellular last mile. The seamless layering of these capabilities is designed to result in an unsurpassed user experience for every user, every time.

Mobile Performance Optimization With Akamai


Customer Origin
Non-Akamaized sites require multiple long-haul roundtrips across both mobile and IP networks
High-loss, high-latensy mobile networks result in numerous retries, retransmits, and other delays.
Content not optimized for the mobile experience exacerbates sluggingness and clogs the already overloaded networks.
IP Network
GGSN
Mobile Operator Network
The Akamai Mobile Protocol improves performance over slow, lossy cellular network
Akamai Mobile Protocol
Akamai deliveres blazing performance for fixed line clients from nearby servers, relying on the Akamai Protocol and SureRoute (Akamai route optimization) to ensure optimized communications with the origin.
Terrestrial
Mobile detection and site delivery from Akamai servers sitting near mobile gateways means users get their first byte and every byte–faster and more reliably.
Akamai FEO enables rich, mobile-optimized sites that maximize each user's device and network capabilities.

Footnote: Gateway GPRS Support Node (GGSN). The place where the public internet and the private cellular infrastructure cross over.

Benefits Of Front-End Optimization With Akamai

Offload the Headache

With Akamai, businesses can enjoy the tremendous benefits of FEO with the virtual flip of a switch. There's no big up-front investment, no on-going maintenance nightmare, no struggle to keep up with the latest browser changes. In fact, there are no changes at all to your Akamaized site. This means your development team can simply focus on its core mission of developing compelling content and functionality, while our high-performance platform delivers it with the speed and responsiveness your users expect.

Rely on a Proven Platform

No other FEO solution can match the proven performance, scalability, and reliability of the Akamai Intelligent Platform. With over 170,000 servers in more than 1,300 networks worldwide, Akamai has the world's most pervasive cloud optimization platform – trusted to accelerate as much as a third of the traffic travelling over the Internet each day. Our FEO capabilities are integrated into this time-tested platform, delivering a solution that is highly secure and backed by our performance and uptime SLAs.

The Best Experience for Every User, Every Time

Akamai's layered, integrated approach to performance means that we are able to apply the right tools at the right time for each site, across nearly every scenario. Our device-targeted front end optimizations work in synergy with our site and application acceleration services to deliver unparalleled speed from end-to-end – in a process that is as seamless to our customers as it is to their end users.

Appendix

For businesses looking to deliver the fastest, most engaging user experiences possible, Ion offers a fully-automated solution for situational performance by including Akamai Front-End Optimization (FEO). Akamai FEO applies optimizations based on sophisticated analysis of the web application as well as real-time conditions specific to the end user's environment, such as browser, device, network speed and usage of third-party services. Akamai Front-End Optimization:

REDUCES REQUESTS

Each client HTTP request and server response combination represents at least one round-trip on the network. Depending on the user's network, device and proximity to the origin server, a single round-trip request can take seconds to complete. A single web page can require dozens of HTTP requests before it can render content, with requests often delaying one another due to a limited number of connections. To reduce round-trips, Akamai FEO uses several techniques to eliminate unnecessary requests such as consolidating multiple CSS and JavaScript files and using new caching features in HTML5.

File Versioning:

Many sites only cache content for a few hours due to concerns about old content used from the cache when new content is available. This practice all but eliminates the enormous performance benefit of caching for users returning to a site later – a very common occurrence. Akamai FEO uses a technique called File Versioning to solve this problem, by giving each file a unique name. If a resource changes, the old file will be not be referenced anymore, so all content can be cached indefinitely.

Adaptive Consolidation:

web pages hold dozens or hundreds of resources, each requiring an HTTP transaction to be fetched from the server. These transactions have considerable overhead, are time consuming, and cannot be done all at once. Consolidation is the technique of combining multiple resources into one. Akamai FEO applies smart consolidation of JavaScript and CSS files, reducing the overall page size and accelerating load time.

Advanced Adaptive Consolidation:

While simple consolidation reduces the number of HTTP requests for first time visitors, it often leads to more bytes downloaded. The same CSS & JavaScript files are included in many different combinations across a website's pages. Each combination is consolidated into a separate package, resulting in the same content being downloaded multiple times. Akamai FEO uses Adaptive Consolidation to fetch multiple files with one request, but cache each resource separately, thus avoiding redundant downloads. The result is the optimal number of requests, without increasing the payload.

Inlining:

Many of the external images and scripts on a page are small. Often so small, that the HTTP overhead to fetch them exceeds their own size. Inlining is the technique of embedding these resources directly into the page or resources that references them. The overall page size decreases with the reduced HTTP overhead, and the inlined resources are processed faster. Akamai FEO automatically embeds resources into pages and CSS files based on size and other considerations.

Advanced JavaScript Inlining

Inlining JavaScript into HTML is almost always faster the first time a page is loaded as the browser doesn't have to make additional requests for JavaScript. However, basic JavaScript inlining can make subsequent page loads slower because the browser must download larger HTML (containing the inlined JS) rather than loading the JavaScript directly from the browser cache. Advanced JavaScript Inlining offers the best of both worlds. On first view JavaScript is inlined and stored in HTML5 localStorage. On the second view the JavaScript is loaded from HTML5 localStorage.

On-demand Image Loading:

Pages are often much larger than the size of the browser window, especially on a small mobile screen. And yet, the browser will load all the images on the page regardless of which ones are visible to the end user. Akamai FEO's On-demand Image Loading optimization will cause a page to only load the images that are visible within the current viewport. As the user scrolls down, new images are loaded on demand. On-demand Image Loading helps improve page load time and also reduces bandwidth for cases where a user doesn't actually scroll down a page.

HTML5 Advanced Cache:

Caching previously downloaded page elements on users' browsers is one of the best ways to improve performance of repeat views. However, since the cache size on mobile browsers is very small, few files last in the cache until the user returns to the site. Even on desktop browsers, users cycle through their cache in days. Akamai FEO cache optimization leverages a new capability in HTML5 to bypass the shared browser cache and create a dedicated cache for each site.

REDUCES BYTES

The math is simple: the larger a web page (measured by bytes), the longer it will take a browser to render the content. Akamai FEO keeps file size in check by adjusting image formats, improving cache management, compressing files, and removing metadata from files, (such as comments, whitespaces and image metadata). Akamai FEO also helps avoid excessively large images, maintaining image quality at the edge of what the user can perceive, on a small and large screen.

Optimizations that contribute to page-size shrinkage include:

Compression:

Compression is an effective way to reduce page sizes and resources without impacting their content. HTML, JavaScript and CSS files are textual and compress very well, as do AJAX responses. Compression can often reduce 60%-80% of the total page size. Akamai FEO automatically compresses all text content, and sometimes even images, while ensuring compressed content is only served to clients that support it.

Lossless Image Compression:

Images often hold significant amounts of metadata, ranging from where a picture was taken to layering of its graphics for easy future editing. In addition, images are often saved in the format most suited for image editing software – not for the web. Image Compression removes metadata and uses the optimal image format for the web, resulting in smaller file sizes with no difference in image quality.

Resize images to HTML dimensions:

HTML and CSS designs are often out of sync with the management of images. Files are often uploaded to a Content Management System without knowing the size they'll be displayed in, and the same image is often used in multiple sizes. Akamai FEO makes sure the browser will not download a large image, just to have it resized by the browser due to styles and markups. Such images will be resized ahead of time, avoiding downloading of redundant bytes.

Browser-Specific Image Optimizations - WebP (Chrome/Opera), JPEG XR (IE9+) & JPEG2000

New browser-specific image formats allow the same quality of experience to be delivered in a reduced payload size, when compared to standard JPEG. Akamai FEO recognizes situations where JPEG images can be replaced by browser-specific image formats resulting in a significantly better user experience.

Minification:

HTML, JavaScript and CSS files contain comments and whitespace that, while useful to developers, are not needed for the page's operation. Minification is the process of removing such components and reducing the total download size. Akamai FEO automatically minifies all page resources, reducing the size of pages without modifying their functionality.

Responsive Images:

Most images are formatted for desktop browsers on larger screens. Such images are unnecessarily large for mobile devices. Akamai FEO's Responsive Images technology automatically delivers an optimally-sized image for the phone, tablet or desktop browser that requested it, resulting in much faster page load with no perceived difference to the user.

Intelligent Image Quality:

The improvements in digital photography often lead to web page images saved at a higher pixel density than what most displays can support. Such images make the pages bigger and thus hurt the user experience, while the user cannot appreciate the higher quality picture. Akamai FEO optimizes the pixel density to reduce size without a perceptible impact on quality.

ACCELERATES RENDERING

Processing a web page is a complicated process. Browsers employ complex logic during load time, making decisions such as which files to download serially vs. in parallel, which resource types block rendering, and how to manage their connections. At the same time, they need to parse and execute complicated HTML, CSS and JavaScript code, which is often not well defined.

Unfortunately, the browser doesn't know sites in advance and is forced to employ generic logic when processing a page. This logic changes between old and new browsers, is limited by backward compatibility, and is not customized to a site.

Akamai FEO combines the knowledge it has of a site–learned by automated analysis, inspection and user configuration–to identify the best way to load that page. Techniques like deferring print stylesheets, keeping social buttons from blocking rendering and prefetching the next page are used to guide the browsers into doing the right thing. As a result, users can get a truly fast user experience, attuned to their needs.

Optimizations that contribute to accelerate rendering include the following:

EdgeStart:

EdgeStart reduces the time to deliver the first part of the HTML response, which allows the browser to download important resources such as JS, CSS and some images earlier to enhance the user experience. This optimization can be particularly helpful for mobile sites and for end users distant from the origin. EdgeStart leverages Akamai's global footprint of Edge servers to get a response back to the end user as quickly as possible.

Asynchronous JavaScript and CSS:

When processing a page, browsers usually process JavaScript and CSS files as blocking resources. This means images and other resources further down the page aren't downloaded until the JavaScript and CSS files are fully downloaded and executed. Even rendering is delayed until all CSS and preceding JavaScript is fetched and run. Akamai FEO's Async JavaScript and CSS processing optimization modify the way scripts and style sheets are embedded into the page, making the browser process scripts, style sheets and other resources in parallel. This improves load time and makes the page start rendering much earlier, while still not showing unstyled content.

Streaming Consolidation:

If you consolidate 10 JavaScript files into one, you make the first one significantly slower. When downloading JavaScript and CSS, browsers do not process the content until the entire file is downloaded. As a result, consolidating resources, while it reduces HTTP requests, sometimes makes the page slower. Akamai FEO combines IFrames and its Async optimizations to overcome this problem, processing incoming JavaScript and CSS as it arrives, without additional HTTP requests.

Invoke Click On-Touch:

Most touch screen browsers fire an on-touch event roughly 300 milliseconds before firing an on-click event, in order to distinguish between pinch, zoom and other touch operations. Akamai FEO's Invoke Click On-Touch leverages the on-touch event to simulate fast clicks, even on a touchscreen, resulting in a significantly more responsive page. This method applies to iPhone, iPad, and Android devices.

JavaScript Pre-Execution:

Executing JavaScript on mobile devices can be slow. On average, mobile browsers take ten times longer than desktop browsers to process JavaScript. When JavaScript Pre-Execution is enabled, Akamai FEO executes much of the embedded JavaScript offline and provides the requesting browser with a mostly static page. Even fast-changing scripts and interaction scripts are made faster by "statifying" other scripts which generate content. Using this method reduces the amount of JavaScript that either the client or browser will need to execute at page-request time, lessening some of the mobile device's processing burden.

Page Prefetching:

This optimization loads page content ahead of time based on users' browsing patterns. For example, say 60% of visitors to a home page click the "Hot Deals" link. While a visitor views that home page, Akamai FEO can hint to the browser to load the "Hot Deals" page without waiting for the user to request it. Then when the user does click the link, the "Hot Deals" page will display immediately.

HTML5 Resource Prefetching:

Resource Prefetching aims to reduce the total time it takes to navigate multiple pages across a website. When enabled, this optimization allows a page to download the JavaScript and CSS resources from other configured policies. When the user accesses those future pages, the resources will already be stored in HTML5 local storage providing a faster overall experience.

DNS Prefetching:

In modern web pages, there are often dozens of calls made for content on external domains. Performing a DNS resolution for each individual hostname does not take long, but with enough resources served from external domains this can lead to a performance bottleneck. DNS prefetching bypasses this by performing the DNS resolutions before objects on external domains are actually requested.

Image Placeholders:

On-demand Image Loading downloads images as they are needed by end users. Image Placeholders improves the user scrolling experiences by using lower resolution placeholder images until the placeholders come into view and the full quality images are needed.

Akamai continually updates and improves its FEO optimization engine to reflect the state-of-the-art in FEO innovations as well as the constantly evolving browser and device landscape. This enables our customers' development teams to focus simply on creating innovative content and features, rather than getting bogged down by the ever-changing details around the latest FEO optimizations and which versions of which browsers support what features.

Domain Sharding:

Browsers impose limits on the number of parallel connections per hostnames. Although with modern browsers these limits tend to be adequately high, older browsers can be restrictive. This leads to artificial dependencies as the browser waits for otherwise independent objects to be fully delivered before proceeding with further requests. Domain sharding gets around this limit by distributing requests over multiple subdomains, ensuring that the browser does not hit the limit of parallel requests per hostname.

[1] http://www.websiteoptimization.com/speed/tweak/average-web-page/

[2] http://www.gomez.com/pdfs/wp_why_web_performance_matters.pdf

[3] "For Impatient Web Users, an Eye Blink is Just Too Long to Wait", New York Times, February 29, 2012

[4] http://www.stevesouders.com/blog/2012/02/10/the-performance-golden-rule/

[5] http://www.websiteoptimization.com/speed/tweak/average-web-page/

[6] http://www.compuware.com/d/release/592528/new-study-reveals-the-mobile-web-disappoints-global-consumers

Exhibit V[1]

[1] Downloaded from: https://techdocs.akamai.com/ion/docs/front-end-optimization-ion

Front-End Optimization User Guide for FEO Standard and FEO Premier

February 10, 2022

Contents

FEO: What You Need to Know

Front-End Optimization is built on our Intelligent Platform™—a network of tens of thousands of servers in hundreds of networks around the world, controlled by systems that route requests, balance load, and ensure uptime.

Important: These FEO features and settings will reach end of life in March 2021:

- *Cellular Connection Keep-Alive*
- *Critical CSS for Mobile*
- *Invoke Click On-Touch*
- *JavaScript Pre-Execution*
- *Page Prefetching*
- *On-Demand Image Loading->Low-Quality Image Preload*
- *Resource Prefetching*

You should disable these options before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations.

Review these topics to get a high-level understanding of how FEO works.

About FEO

Front-End Optimization accelerates web pages by creating modified versions of the HTML and supporting resources that can render faster.

The techniques FEO uses to accelerate pages are referred to as *optimization methods* (or just *optimizations*). Some optimization methods are specific to resource files such as images, JavaScript, and style sheets. Other optimizations adjust the prioritization and queuing of rendering activities; these methods include Asynchronous JavaScript, DNS Prefetching, and On-Demand Image Loading.

In short, FEO takes web pages from the origin server, then optimizes and delivers a modified copy of the content to the requesting browser. The versions of resources that FEO has changed are considered *optimized*. We may also refer to such content as *transformed*.

There are dozens of specific optimizations, which do one or more of the following:

- reduce the number and size of HTTP round-trip requests that a page requires to load.
- pare down the size of the page and/or its resources by using more efficient formats, improving caching, and removing unnecessary content.

- accelerate page rendering by adjusting the order and timing with which the browser loads page objects.

ⓘ **Note:** With FEO, no changes are made to the content on your origin.

ⓘ **Note:** FEO does not affect Search Engine Optimization (SEO) results in any way. When a web-crawling user agent requests a page that is enabled for FEO, Akamai returns the origin page, not the transformed page.

Basic Concepts & Terminology

FEO uses language in a very precise way to conceptualize features, behaviors, and interactions with other technologies. Before getting started with FEO, review the list of terms below.

Terms are organized around three contexts: 1) Configuring Akamai services, 2) configuring FEO in the Akamai Control Center, and 3) implementing FEO, which happens behind the scenes on the edge network.

Terminology for Service Configuration

If this is the first time you are configuring Akamai services, review the following table to become familiar with the terminology used for the host and origin processes.

Service Implementation Terminology

Term	Description
digital property	The digital property is an Akamai-assigned identifier that indicates which configuration file and application to use when processing requests. Often, it is the full, end-user-facing hostname or domain name of your application—for example, `www.example.com`.
origin server	The origin server is where you upload your content for users to access via your web site. This is typically maps to the URL the user employs to link to your content.
origin server's *expected* hostname, or forward HOST header	The expected hostname, sometimes known as the Forward HOST header, is the name the edge server forwards to the origin in the HTTP HOST request header. Often, this is the same name as the digital property.

Terminology for FEO Configuration

Terms	Definition
FEO Configuration	For FEO, the collection of optimization policies assigned to a single property constitutes a configuration. As is the case for other Akamai products, a configuration is represented by an

Terms	Definition
	auto-generated XML file containing the metadata Akamai uses to deliver your content.
Parent Product Configuration	A configuration in Property Manager for the product that provides you with the FEO feature, such as Ion, Dynamic Site Accelerator, or Rich Media Accelerator. To configure FEO, you first enable the FEO behavior in your parent product configuration
FEO Resources Configuration	Another configuration in Property Manager using your parent product. Its purpose is to manage delivery of FEO-optimized resources. You create a dedicated FEO resources property that identifies your FEO NetStorage group and the edge hostnames for your domain shards.
Page Type	A set of pages that perform the same function on you site and share similar elements such as CSS files and JavaScript. For example, an e-commerce site might consider all of their product detail pages as a separate page type than their product line overview pages, which simply list the different products.
Optimization Policies	**Optimization policies have two components: match criteria and optimization methods.** Optimization policies determine which pages to optimize and which optimization methods to apply to the qualifying pages' content. Optimization policies should be built around a specific page type. This makes it easier to apply the same set of optimizations to pages that are structurally similar. For example, an e-commerce site might use a policy entitled *ProductPages* that would match on all product detail pages and use optimization methods such as Image Optimization, Asynchronous JavaScript, and CSS Minification.
Match Criteria	There are two types of match criteria: URL matches and browser matches. URL matches identify HTML pages sharing certain characteristics such as HTML structure or resources. You use URL matches, either in the form of wildcards or regular expressions, to associate page URLs with optimization policies. Browser matches allow you to apply policies based on the requesting browser. A policy will apply only to requests from the selected browsers;

Terms	Definition
	requests from other browsers are excluded from the policy.
Optimization Methods	An optimization method is a specific technique that FEO uses to improve page performance. Typically, you would use multiple methods to optimize your site as each method performs a different function. Some methods are dedicated to a particular content type; other methods manage broader processes such as HTTP requests.
Exclusions	An exclusion shields discretely-defined content from a single optimization method. For example, you might have third-party content on your site that you must not alter in any way; so applying image optimizations would be problematic. In such a case, you could construct a policy that optimizes all images on your pages *except* those originating from an external source. There are two types of exclusions: **policy** and **global**. Policy-level exclusions apply individually. Global exclusions prevent application of an optimization method to all of the policies in the configuration.
URL Constant	If, despite meeting a policy's URL matches, the first-requested URL is not structurally similar to other pages in the policy, then the FEO Analyzer may generate inefficient transformation rules. To override the default usage of the first URL as the benchmark for all policy transformations, enter a URL Constant. FEO Analyzer will then ignore the first-requested URL and instead use the URL Constant as the model for all transformations for all pages in the policy.
Suggested Policy	Also referred to as a Recommended Policy. To get a suggested policy, you submit the URL of a page you want optimized. FEO's analysis engine returns a collection of optimization methods as a preconfigured policy that you may then add to your FEO configuration.
Resource Delivery	Resource Delivery settings are key pieces of information needed for the Intelligent Platform to serve FEO content to end users. It consists of two main elements:

Terms	Definition
	• how to use your NetStorage account to store optimized resources, and • whether to deliver resources via domain sharding or a virtual path.
NetStorage	An Akamai service that provides persistent, replicated storage of site content. It mirrors content to core network locations, which makes the content easily accessible to edge servers. You create a dedicated NetStorage group for your FEO-optimized content.

Terminology for FEO Implementation

Terms	Definition
First-Requested URL	The first time a browser requests a URL that maps to a policy, for example Policy A, FEO uses *this* URL's page content as the basis for optimizing all pages in Policy A. Thus the first-requested URL becomes the policy benchmark.
FEO Analyzer	The FEO Analyzer is an FEO component working on the back end. It analyzes the first-requested URL to determine which optimizations should apply to all of the pages in the policy. The Analyzer produces instructional analysis rules, containing two pieces: • A list of **transformations** (the actual HTML changes required to optimize the pages), and • A corresponding list of which resources to transform.
FEO Transformer	Modifies a copy of the origin's resources based on Analyzer's instructions. Once transformed, these resources are optimized and cached in the edge network. The original files, or base resources, stay untouched and unoptimized on the origin server.
Analysis Rules	The set of transformation instructions the Analyzer creates and gives to the Transformer. The instructions include which resources to optimize and how to do so.
Page	FEO measures a page as a single unit of text or HTML rendered by a specific URL. A page's URL, by definition, must not resolve to any other content

Terms	Definition
	type such as an image (.png, .jpeg, etc.) or unit of media (.mp4, .flv, etc.).
Source Page (Origin Page)	The source, or *origin*, page is the initial content on the origin server that has not been modified by FEO. FEO uses the content from an origin page as a base, but never changes the content on the actual origin.
Optimized Page (Transformed Page)	FEO copies the content from the origin, optimizes the page's HTML, and serves an optimized page, sometimes called a *transformed* page. The optimized page is comprised of the content resulting from applying the optimization methods dictated by the FEO configuration.
Transformations	The altered code snippets that FEO has injected into the HTML of optimized pages.

Key Features

Front-End Optimization (FEO) accelerates web pages by modifying the pages' resources and HTML. By automatically streamlining the code on a web page during the delivery process, FEO expedites the transmission and rendering of the page's content. As a cloud-based service that requires no user-implemented code, FEO is designed to work with any web site.

There are dozens of specific optimizations, which do one or more of the following:

- Reduce the number and size of HTTP round-trip requests that a page requires to load.
- Pare down the size of the page and/or its resources by using more efficient formats, improving caching, and removing unnecessary content.
- Accelerate page rendering by adjusting the order and timing with which the browser loads page objects.

See the table to get more details about these features and capabilities.

FEO features, described

Feature	Description
Reducing round-trip requests	• Each HTTP client request and server response combination equals one round-trip on the network. Depending on the speed of the requesting device and the user's proximity to the origin server, a single round-trip request can take a second or longer to complete. A single web page can require dozens of HTTP requests before it can render content.

Feature	Description
	• To reduce round-trip requests, FEO uses several request-limiting techniques such as eliminating unnecessary requests, consolidating multiple style sheets and JavaScript files, and using new caching features in HTML5. • Optimization methods that contribute to request reductions include the following: – File Versioning – Asynchronous JavaScript – Optimize CSS – Optimize JavaScript – On-Demand Image Loading
Paring down page/resource size	• The math is simple: The larger a web page (measured in bytes), the longer it will take a browser to render the content. FEO keeps file size in check by adjusting image formats, improving cache management, compressing files, and consolidating code in style sheets and in JavaScript files. The process of file consolidation is called *minification*. It strips JS, CSS, and HTML files of the visual cues humans need—such as line breaks, spaces, developer comments, and long-form syntax—and returns source code that allows a browser to more quickly generate a visually identical web page. • Optimization methods that help pare down the page size include the following: – Optimize Images – Optimize CSS – Optimize JavaScript
Accelerating page rendering	• The default behavior of front-end processes—such as sending HTTP requests, running scripts, downloading resources, and displaying human-readable

Feature	Description
	content—is to proceed serially. That is, the browser performs one task at a time and postpones each task's launch until the preceding task is complete. • Task order is determined by a range of factors, including where in the HTML page the task's code is located. Because many pages define style sheets and JavaScript dependencies near the top of the source-code page, these elements can delay content download and display. FEO circumvents many of the delays by running more processes in parallel and, when sequential activities are necessary, reprioritizing content-generating tasks. • Optimization methods that contribute to streamlining processes include the following: – Asynchronous JavaScript – Defer Print Style Sheets – JavaScript Pre-Execution – Page Prefetching

How FEO Works: A High-Level Breakdown

Initially, it is the browser’s request for content from an Akamaized web site that triggers the FEO process. Here is a simplified sequence of events from the initial request to the delivery of FEO-optimized web content.

How to

1. A browser sends an HTTP request to an Akamaized web site.
2. Akamai determines that the site is provisioned for FEO and then proceeds to check for an active FEO-specific configuration file. The FEO configuration specifies which optimization methods to apply and under what circumstances.
3. Assuming the presence of an active FEO configuration, FEO then optimizes the requested content and caches it on edge servers.
4. Subsequently, additional HTTP requests for the same content will receive the cached, optimized content.
5. After the expiration of the Time-to-Live (TTL) (or Max Age) setting in your parent product’s configuration, one of two things will happen:

- If there is an active FEO configuration file, FEO will repeat the optimization process upon the next HTTP request, and it will return freshly-optimized content.
- If there is no longer an active configuration, the FEO-optimized content expires and the requesting browser will receive the unoptimized content retrieved from the origin server.

Configuration Types Explained

To use FEO, you will need three configurations:

- a parent product configuration,
- an FEO resources configuration, and
- an FEO configuration.

These configurations do not all use the same configuration tool. You access the parent product and FEO resources configurations through Property Manager. You access the FEO configuration through FEO Configuration Manager.

Parent Product Configuration

The parent product configuration controls the function of multiple Akamai features. You should be familiar with the existing configuration(s) for your parent product.

The key change you will make to your parent product’s configuration is enabling FEO. This only makes the FEO feature available to you. It does not prompt FEO to perform any analysis or optimization.

ⓘ **Note:** You must enable FEO in the parent product configuration first. FEO configurations do not work until you have completed this step.

Generally, FEO does not affect how your parent product’s features work, with the obvious exception that some resources delivered will be optimized. However, there are certain combinations of settings that are not compatible between FEO and the parent product. The FEO user documentation flags these combinations.

FEO Resources Configuration

You also need a configuration in Property Manager that controls how optimized resources are sent to edge servers. This configuration is for a separate FEO resources property that is on the same account as your parent product configuration.

One function of the FEO resources configuration is to connect FEO to your FEO NetStorage group. NetStorage is an Akamai service that provides persistent, replicated storage of site content. It mirrors content to a small number of core network locations, which makes the content easily accessible to edge servers. You create a dedicated NetStorage group for your FEO-optimized content. In the FEO resources configuration, you tell the Akamai network to use NetStorage as the origin for your optimized content.

The FEO resources configuration also helps support domain sharding, a technique for speeding load times by creating more connections to client browsers. In the FEO resources configuration, you create an edge hostname for each domain shard.

For more information about resource delivery, NetStorage, and domain sharding, see the following:

- *FEO and Domain Sharding*
- *Considerations for Domain Sharding*
- *Configuring Domain Sharding for HTTP Requests*
- *Configuring Resource Delivery for HTTPS Requests*

FEO Configuration

Finally, the FEO configuration provides the information necessary for FEO to analyze your site's content, perform optimizations, and deliver the optimized resources. FEO configurations function much like your parent product configurations. Both types of configurations can be created and modified using an interface in the Akamai Control Center that will auto-generate metadata in the correct XML format. There are two key differences:

- The configuration for your parent product controls settings for multiple Akamai products and services, whereas the FEO configuration defines the behavior for Front-End-Optimization only.
- Instead of Property Manager, you use a different tool to create and manage FEO configurations: the FEO Configuration Manager.

ⓘ **Note:** Creating an FEO configuration does not start optimization. FEO will not optimize any resources until you activate an FEO configuration.

A Closer Look at FEO Configurations

As is the case with other Akamai products, you manage your FEO implementation using configuration files in the form of metadata, stored as XML. FEO's configuration files work somewhat differently than other Akamai configuration files. For one thing, FEO has a dedicated configuration file that must be accessed and maintained in an FEO-only section of the Akamai Control Center. Also, constructing FEO configurations entails different steps. Consider the main elements that comprise an FEO configuration:

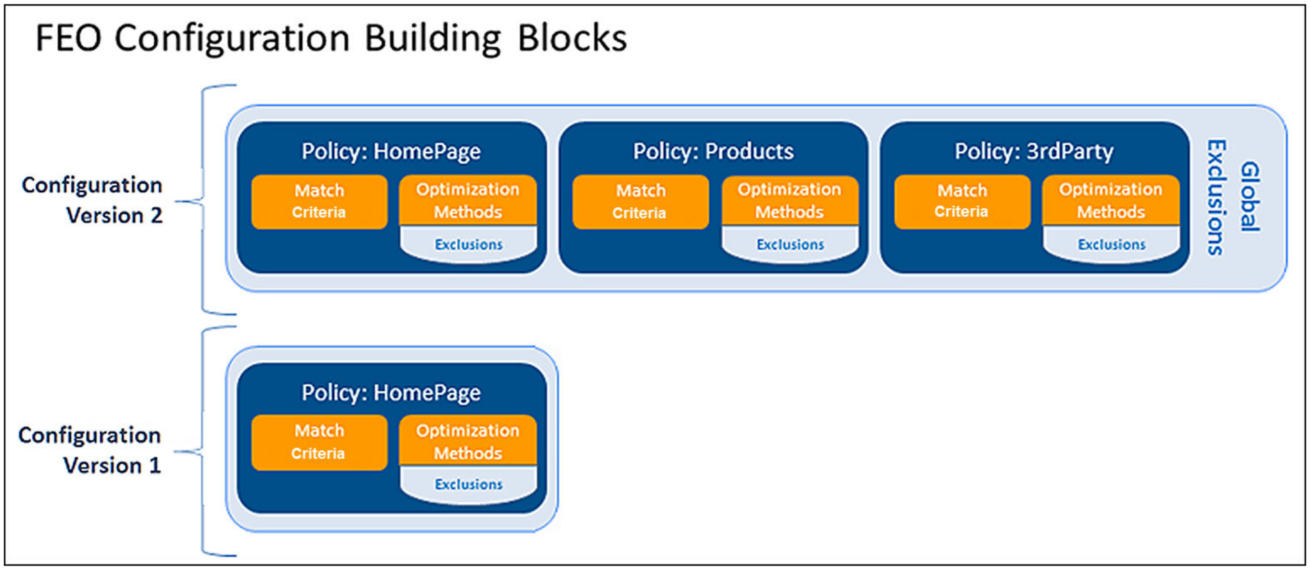
- An **FEO configuration** contains optimization policies, global exclusions, and resource delivery settings; a configuration can have one or more policies and zero or more global exclusions.
- An **optimization policy** represents a collection of optimization methods to apply to a set of pages that share similar content, function, and structure.
- An **optimization method** is the actual technique FEO uses to optimize web content.
- Individual optimization methods may have policy-level **exclusions**, which protect specific content from that particular optimization.
- A configuration can have zero or more **global exclusions** to prevent application of an optimization method to *all* of the policies in the configuration.
- Each FEO configuration includes exactly one set of **resource delivery settings**; they include information about the location that delivers resources (NetStorage) and the method of delivery (domain sharding or virtual path) for HTTP and HTTPS requests.

ⓘ **Note:** For more information about these terms, see *Basic Concepts & Terminology*.

Take a look at the figure "FEO's hierarchy of configuration elements." The diagram shows two possible versions of a configuration file, with the second more complicated than the first. (Because the Akamai Control Center displays configuration files with the newest at the top of the list, this diagram uses the same sorting method.)

1. The first configuration version is simple: It contains one policy. This policy is supposed to optimize the property's home page, so it uses URL matches to detect HTTP requests for the home page. The HomePage policy includes optimization methods selected for their relevancy to the home page's content. To protect some content from changes, individual methods have local exclusions defined.

2. There are two key differences between the versions: First, Version 2 contains three policies, each named to reflect the target content. Second, with the addition of two new policies, it may be necessary to define global exclusions that will protect content from all of them. So Version 2 has global exclusions applied, whereas Version 1 does not.



FEO's hierarchy of configuration elements

The State of FEO Configurations

Getting started with FEO is a two-step process: first, you must enable FEO as an optional feature of your parent product; then you must create, define, and activate an FEO Configuration. We use the terms enable/disable and activate/deactivate to describe different states of FEO configurations. FEO can be enabled, but not active; to be active it must also be enabled.

The *Configuration History* page indicates if FEO is enabled and which configurations are active on the production and staging environments.

The distinctions between enabled/disabled and activated/deactivated depends on status information from two kinds of configurations: the FEO configuration and the parent product configuration. Using FEO also requires an FEO resources configuration (see *Configuration Types Explained*).

A high-level description of each FEO state, coupled with its controlling configuration, follows.

Determinative Configuration	State of FEO	Description
Parent Product	Enabled	FEO is enabled, for a specific account, as an optional feature of the parent product. The account's web content is eligible for FEO.
	Disabled	The account's configuration metadata does not indicate FEO is enabled; when Akamai serves content for the account, the parent product will not run FEO optimizations.
FEO	Active	FEO is enabled for the account as an optional feature of the parent product, and an FEO Configuration Version has been pushed to the network. A version can be active on the Edge Staging Network (ESN), Production, or both.
	Inactive	An inactive FEO Configuration Version was at one point active on one or more networks, but no longer determines how FEO will be implemented for the associated property. Inactive configuration versions can be reactivated.

How FEO is Like an Appliance

Think about FEO as an appliance with both a power cord and a power button. When FEO is enabled, its power cord is plugged in. A plugged-in appliance is ready to use power; but this does not mean it is doing so. Similarly, when FEO is enabled, it is ready to be used but still requires an active FEO configuration to do anything. Activating an FEO configuration version (pushing it to a network) is like turning on an appliance by pressing the power button.

Additional Resources

To get the most out of FEO, refer to additional resources, including parent product documentation and OPEN APIs.

FEO does not stand on its own. It is included as either a standard or optional feature in Ion, Dynamic Site Accelerator (DSA) & Dynamic Site Accelerator Premium (DSAP), Rich Media Accelerator (RMA), Terra Alta, and Web Application Accelerator (WAA). Whichever product you use in conjunction with FEO is your FEO *parent product*. You should be familiar with your parent product and its configurations on your property.

The FEO documentation is a companion reference to the documentation for your parent product. User guides, implementation guides, and online help typically include information about managing your domain name, Edge Hostname, origin server, SSL certificates, caching preferences, and related tasks required before you can successfully use Akamai products. For example, if you are using FEO with Ion, you will want to refer to the Ion documentation; if you are using Rich Media Accelerator, you will need *its* documentation. And so forth.

Consult your parent product documentation if you have not yet read it or if you need a refresher on enabling FEO for use with other Akamai products.

You can access product-specific documentation from the Akamai Control Centerby selecting the **Support** link at the top of any page and then selecting **User and Developer Guides**. Additionally, you may look up topics in the knowledge base by using the search tool, which appears at the top right-hand corner of every Control Center page.

Also, some FEO functionality can also be used via our Open APIs. Refer to *the API Docs* for more information.

When to Escalate to Customer Support

If you are experiencing an issue related to Front-End Optimization, you should first look for a solution in the documentation.

Begin by reviewing the following sections of this online help:

- *Starting the FEO Setup Process*
- *Using FEO's Configuration Manager* on page 31
- *Optimization Methods* on page 47

Also, search the Knowledge Base and Community site for the issue.

If you still cannot find a solution to your FEO issue, enter a support ticket. When entering a ticket, be sure to include the following data to aid the troubleshooting process:

- URL(s) affected
- Digital property in use
- Symptoms of the issue (include a screen shot if possible)
- Browser(s) displaying the issue
- Steps to reproduce
- The HTML source code from both the origin source and optimized pages
- User agent(s) from logs, if available

Considerations

Before you configure FEO, familiarize yourself with certain special considerations. This includes interactions between FEO and other Akamai features for images and other content, domain sharding, HTTP/2, and PCI compliance.

Interactions with Image Settings in the Parent Product

The main concern with using multiple features on images is how to get the desired image quality.

There are three parent-product features that can change an image's resolution:

- Adaptive Image Compression (AIC)
- Image Converter (IC)
- Image Manager

AIC and IC apply only to images that FEO has *not* already optimized. (For example, images with a TTL of less than two hours would be optimized by AIC.) This ensures that no images will be double-compressed. Keep in mind that *any* IC commands will be skipped for FEO-optimized images, even commands that do not involve image compression. Therefore, you might want to put the FEO rule last in your parent configuration to reflect the effective results.

Also, FEO optimizes *only* the images on the analyzed pages for each of your policies. For example, if one of your policies analyzes the contents of the first of three pages in your product catalog, FEO will optimize your company logo, the seasonal sale banner image, and all of the thumbnails for the products on that page. When users advance to the second page, however, only the logo and sale banner images will be optimized. FEO will *not* optimize the product thumbnails on that page.

Image Manager is *not* compatible with FEO optimizations that affect image quality. You can use any other FEO optimizations with Image Manager.

Here are some recommended best practices regarding FEO, AIC, IC, and Image Manager:

- If you wish to use IC commands—such as resizing or cropping—be sure FEO does not optimize those images. No IC commands are applied to FEO-optimized images.
- Add an AIC rule to your parent-product configuration that will catch any JPEG files that are not on FEO-analyzed pages.
- Both AIC and IC will skip compression for any images that FEO has already optimized.
- Do not apply Image Manager to any resources that use Optimize Images or image-specific settings in Optimize CSS.
- The **On-Demand Image Loading** optimization method (also known as Just-in-Time) works with AIC, and can facilitate performance gains if a lot of images are off screen.
- If image zooming is important for an application, image optimizations of any kind might not work well. If you apply optimizations to such images, be sure to test the results carefully.

Tips for Using FEO Image Optimizations

FEO has a few different options that optimize images. These tips can help ensure that images on your site are optimized as you intend.

- **Check each policy's logic carefully.**Confirm that the intended optimization and settings are applied to each image. The **Test URL** feature can help you identify which policy applies to a particular page and its images.
- **FEO optimizes *only* the images on the analyzed page for each policy.**For example, if one of your policies analyzes the contents of the first of three pages in your product catalog, FEO will optimize your company logo, the seasonal sale banner image, and all of the thumbnails for the products on that page. When users advance to the second page, however, only the logo and sale banner images will be optimized. FEO will not optimize the product thumbnails on that page. You can determine which page FEO analyzes for a policy by setting a URL Constant.
- **Some images on your site might be part of the CSS, not the HTML.**To optimize these images, add the Optimize CSS method and select the appropriate settings for optimizing images.
- **FEO handles optimizations intelligently to achieve performance gains.**If optimizing an image would not sufficiently improve performance, FEO might not apply the optimization to that image. For example, suppose a policy tells FEO to convert images to WebP format for requests from Chrome browsers. If FEO determines that a particular image file would be about the same size in WebP format, it will skip that file and only convert the images where the WebP format file size is smaller.

How Other Akamai Features May Affect Your FEO Optimizations

In addition to features that affect images, some Akamai features can interact with FEO optimizations for images and other content types. If your parent product configuration includes any of the features listed in this section, please review the associated FEO considerations.

For information about how Akamai features related to images might affect your FEO configuration, see *Interactions with Image Settings in the Parent Product* on page 16 and *Tips for Using FEO Image Optimizations* on page 17.

The following table provides information on other Akamai features that may affect your FEO optimizations.

Feature	FEO Considerations
Web Application Firewall (WAF)	WAF configurations may block requests from the FEO Analyzer, if geographic blocking is used. Contact your account representative if you use geographic blocking with WAF.
Global Search & Replace	If you use hostname rewriting with the Global Search & Replace feature, any hostname rewrites are applied *after* FEO transformations. Given this, if any of your FEO URL matches include hard-coded links, the FEO optimizations will not be applied.
Edge Side Includes (ESI)	If ESI is enabled on the property-level configuration, your FEO optimizations may not be

Feature	FEO Considerations
	applied because ESI transformations are applied first. If you use ESI, contact your account representative for a workaround.
Device Characterization (DC)	DC provides device characteristics information to the origin. Based on this information, the origin can provide a different version of an object. (For example, an end user on an iPhone can receive a custom HTML object for iPhones.) If you use DC, contact your account representative about your options when using FEO and DC in tandem.
Caching	FEO currently ignores base-page caching. If you use base-page caching, contact your account representative for a workaround. For resource caching, FEO takes into account the TTL of the objects: If the TTL of the resource is less than 2 hours, it won't be optimized by default.
Shopper Prioritization Application (SPA)	SPA may affect downstream caching, which may also prevent FEO from optimizing any of the site's resources. To work around this potential issue, cache only the base page, not the resources.

Considerations for Domain Sharding

When FEO optimizes the resources for a page—images, JavaScript files, style sheets, etc.—it stores them in a NetStorage group that is specific to FEO, ready to be retrieved when FEO-optimized pages need them. You must decide how FEO delivers these resources. One option is *domain sharding*.

Review *FEO and Domain Sharding* to learn what domain sharding is and how it works in FEO.

For configuration instructions, see:

- *Configuring Domain Sharding for HTTP Requests*
- *Configuring Resource Delivery for HTTPS Requests*

Considerations for HTTPS

Domain sharding can be very effective at improving load times for HTTP requests. But for HTTPS requests, it is often not recommended. Here's why.

When an end user requests a resource using HTTPS, the browser and the domain exchange handshakes to establish the secure connection. If there are multiple domain shards (e.g., feo1.example.com and feo2.example.com), the browser must exchange handshakes with *each* domain. Often, the time required to establish multiple secure connections to additional domains is greater than the time saved by increasing the number of connections available for loading resources simultaneously.

Take this into consideration when deciding how to deliver resources for HTTPS requests.

Resource Delivery for HTTPS requests is configured on the Resource Delivery tab of the FEO Configuration Details page, in the third pane (**Configure Resource Delivery for HTTPS Requests**).

If you do not want to use domain sharding for HTTPS requests, you have two options:

- Disable HTTPS resource delivery. If you select this option, no FEO-optimized resources will be available for HTTPS requests.
- Bypass domain sharding for HTTPS requests by configuring a virtual path. This ensures that FEO-optimized resources are served for HTTPS requests.

The virtual path option is recommended in most cases. This method changes both how FEO rewrites URLs and how the parent product handles requests for optimized resources.

Instead of inserting a shard number into the URL, FEO replaces everything before the filename with a virtual path, such as `/feo-cdn` . For example, consider a request that FEO might rewrite like this for HTTP domain sharding:

```
http://feo2.example.com/JC89923NV908.jpg
```

For HTTPS, you would configure FEO Resource Delivery to rewrite that URL like this:

```
/feo-cdn/JC89923NV908.jpg
```

The directory `/feo-cdn` does not actually exist. This path functions as a flag to indicate requests that are not receiving optimized content via the domain shards. You will modify your parent product's configuration to find requests that have this flag and serve optimized content from NetStorage.

Preparation for Domain Sharding

If you plan to use domain sharding, begin by deciding what the sharding edge hostnames will be.

Usually, sharding edge hostnames have four elements.

1. A prefix, such as `http://feo` .
2. The shard number (`1` and `2` , if you have two shards).
3. The property's main domain name, such as example.com.
4. The domain that Akamaicontrols for directing traffic through the Intelligent Platform:
   a. `edgesuite.net` if the Secure option is enabled in your parent product configuration
   b. `edgekey.net` if the Secure option is not enabled in your parent production configuration

So if your parent product configuration uses edgesuite.net edge hostnames and you want to configure two shards, your sharding edge hostnames would look like this:

```
http://feo1.example.com.edgesuite.net
http://feo2.example.com.edgesuite.net
```

Similarly, if your parent product configuration uses edgekey.net edge hostnames, your two sharding edge hostnames would look like this:

```
http://feo1.example.com.edgekey.net
http://feo2.example.com.edgekey.net
```

Decide what your sharding edge hostnames will be and make a note of them. You will use them when creating your FEO resources configuration and in the Resource Delivery Settings of your FEO configuration.

FEO and HTTP/2

The next-generation internet protocol (HTTP/2) includes some changes that enhance performance. FEO can detect HTTP/2 requests and responds to them intelligently.

HTTP/2 provides better results than some FEO optimizations. FEO automatically ignores these settings for HTTP/2 requests:

- Optimize CSS
    - HTML5 Advanced Cache: Adaptive Consolidation
    - Small Image Embedding
- Optimize JavaScript
    - HTML5 Advanced Cache: Adaptive Consolidation
    - Inline External JavaScript

PCI Compliance Considerations

⚠ **Caution:** If it is important for your digital property to be PCI-compliant, you must ensure that FEO is not applied to pages that might collect credit card numbers or other card-holder information. Please check that such pages are excluded from your FEO configuration.

Starting the FEO Setup Process

The FEO setup process begins with three essential tasks: enabling FEO, setting up a NetStorage group, and activating an FEO resources property.

The full process of setting up FEO looks like this:

1. Modify the parent product configuration to enable FEO
2. Set up a NetStorage group
3. Create and configure an FEO resources property and activate it
4. Configure FEO settings and policies
5. Activate the parent product and FEO configurations in the testing environment
6. Test FEO functionality
7. Go live

The following sections describe the first three steps to the FEO setup process. Once they are complete, you're ready to configure FEO using FEO's Configuration Manager.

Enabling FEO

Before you can begin configuring FEO, you must first enable it as an optional feature of the configuration for your parent product.

Before you begin

If you have not already done so, review the optional features section of your parent product's implementation guide to learn more.

In most cases, you enable FEO in your parent product configuration by doing the following.

How to

1. Access the Property Home page in Property Manager.
2. Either edit your latest configuration (if it has not yet been activated) or edit a new version of your latest configuration.
3. In the Default Rule, add the **Front-End Optimization** behavior.
4. Set the behavior's **Enable** switch to **On**.
5. Save the edited configuration.

Setting Up NetStorage

These steps are performed in FEO Configuration Manager. In this part, you begin creating your first FEO configuration and generate a NetStorage group to store your optimized resources.

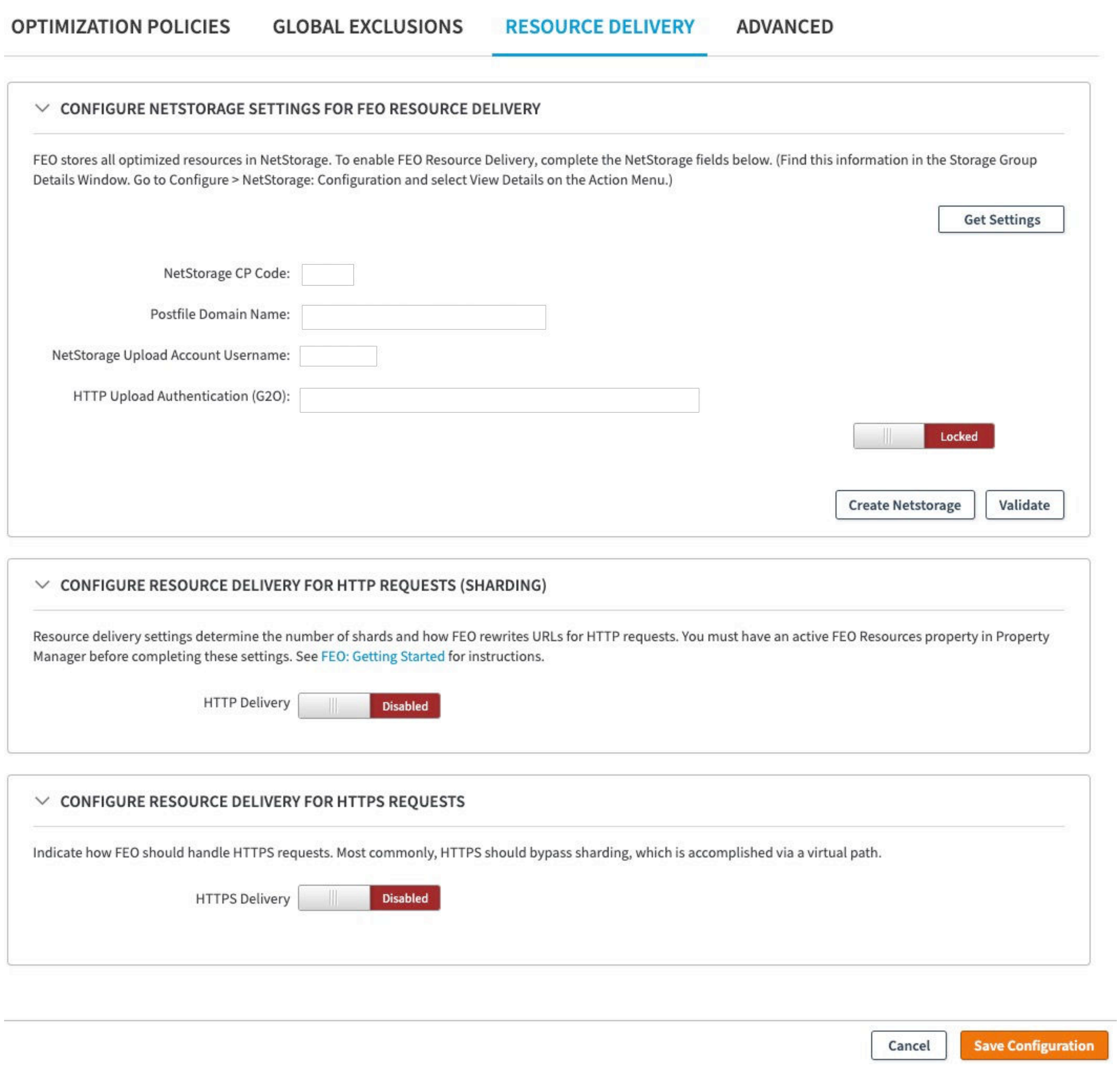
ⓘ **Note:** Creating a NetStorage group requires a user who has administrative permissions on your account.

How to

1. Navigate to the Property Details page for the property you want to configure.
2. On the **Related apps** menu, select **Front-End Optimization**. This displays the FEO Configuration History page.

   💡 **Tip:** If Front-End Optimization does not appear on the **Related apps** menu, check that 1) you are on the Property Details page, not the Account or Group Home page, and 2) FEO is provisioned for and enabled on that property.

3. Click the **Create New Version** button. This displays the FEO Configuration New Version page.
4. Select the **Resource Delivery** tab.



OPTIMIZATION POLICIES | GLOBAL EXCLUSIONS | RESOURCE DELIVERY | ADVANCED

CONFIGURE NETSTORAGE SETTINGS FOR FEO RESOURCE DELIVERY

FEO stores all optimized resources in NetStorage. To enable FEO Resource Delivery, complete the NetStorage fields below. (Find this information in the Storage Group Details Window. Go to Configure > NetStorage: Configuration and select View Details on the Action Menu.)

Get Settings

NetStorage CP Code:

Postfile Domain Name:

NetStorage Upload Account Username:

HTTP Upload Authentication (G2O):

Locked

Create Netstorage | Validate

CONFIGURE RESOURCE DELIVERY FOR HTTP REQUESTS (SHARDING)

Resource delivery settings determine the number of shards and how FEO rewrites URLs for HTTP requests. You must have an active FEO Resources property in Property Manager before completing these settings. See FEO: Getting Started for instructions.

HTTP Delivery: Disabled

CONFIGURE RESOURCE DELIVERY FOR HTTPS REQUESTS

Indicate how FEO should handle HTTPS requests. Most commonly, HTTPS should bypass sharding, which is accomplished via a virtual path.

HTTPS Delivery: Disabled

Cancel | Save Configuration

Resource Delivery Settings

5. In the **Configure NetStorage Settings** pane, click the **Get Settings** button.

   a. If you already have an available FEO NetStorage group, the NetStorage Settings fields auto-populate. Skip ahead to Step 6.

   b. If you do not yet have an FEO NetStorage group, a message appears prompting you to create one.

6. If prompted, click the **Create NetStorage** button.
   In a few moments, the NetStorage Settings fields auto-populate.

7. Make a note of the values in the **NetStorage Upload Account Username** and **NetStorage CP Code** fields.

8. Click the **Save Configuration** button.

A notification message appears reminding you to configure your resource delivery settings before activating. Dismiss this message.

Creating and Activating the FEO Resources Property

You create this configuration in Property Manager. It must be in the same account as your parent product and FEO configurations.

You will perform four tasks: creating the new property, configuring the hostname settings, configuring the rules, and activating the configuration.

Creating the FEO Resources Property

The FEO resources that are created when some FEO optimizations are enabled, need to be served through Property Manager. If you choose to create a separate property to serve FEO resources through separate host names, follow these instructions.

This property must be in the same account as your parent product and FEO configurations.

How to

1. Log in to the Akamai Control Center (*https://control.akamai.com*).
2. Select your account if it is not already displayed.
3. Click the **+Create** button and select the **Property** option.
   a. Set **Property Type** to Property Manager.
   b. For **Product**, select the parent product that includes FEO.
   c. Select the contract for this new property.
4. Enter a name for your FEO resources property. It is recommended that it be similar to your parent product's property name, prefixed with `feo-resources`.

   Example

   Example: `feo-resources.example.com`
5. Click the **Create Property** button.
   The Property Manager Editor page appears.

Next steps

To continue with FEO setup, configure the hostname settings for your FEO resources property.

Configuring Hostname Settings

In this section, you will create an edge hostname for each domain shard. (You will configure the remaining domain sharding settings when you complete your FEO configuration.) The end result is to map each sharding edge hostname to itself. Therefore, you will be configuring the same value in both the Property Hostname field and the Edge Hostname field.

The settings in this task apply to the FEO resources property.

These instructions assume you will be configuring your main domain plus two domain shards. To adapt them, simply create one property hostname entry per domain shard.

If you do not require edgekey.net sharding edge hostnames (i.e., your parent product configuration does not have the Secure option selected), ignore the steps flagged “HTTPS Properties.”

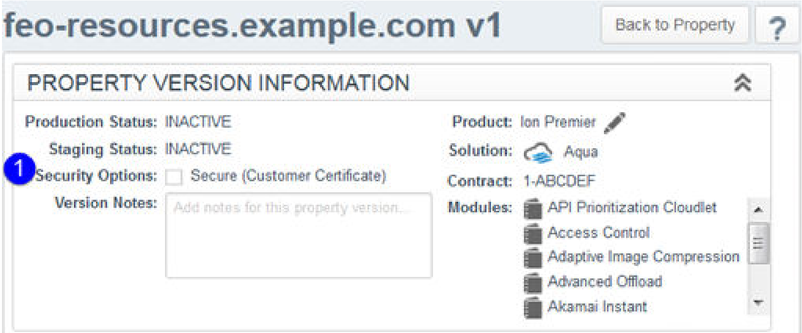
Important: The sharding edge hostnames should be on the same domain as your parent product configuration’s edge hostnames—both should be edgesuite.net, or both should be edgekey.net. Properties that use HTTPS (i.e., the Secure option is selected) use edgekey.net hostnames, and HTTP-only properties use edgesuite.net.

Note:

If your parent product configuration does not have the Secure option enabled, skip the steps that are flagged “HTTPS Properties.”

How to

1. *HTTPS Properties:* Select the Secure checkbox.



feo-resources.example.com v1

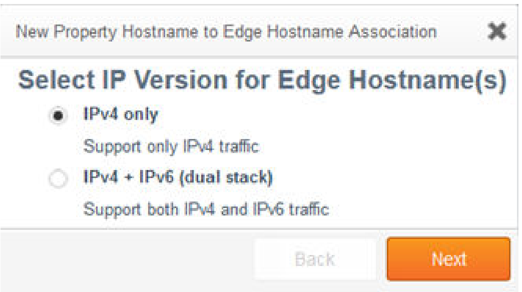
PROPERTY VERSION INFORMATION

Production Status: INACTIVE
Staging Status: INACTIVE
Security Options: □ Secure (Customer Certificate)
Version Notes: Add notes for this property version...

Product: Ion Premier
Solution: Aqua
Contract: 1-ABCDEF
Modules: API Prioritization Cloudlet, Access Control, Adaptive Image Compression, Advanced Offload, Akamai Instant

Secure option (HTTPS-enabled properties only)

2. In the Property Hostnames pane, click the **Add** button.
   The Property Hostnames wizards opens, prompting you to add property hostnames.
3. Enter the sharding edge hostnames you determined earlier, and click **Next**.



New Property Hostname to Edge Hostname Association

Select IP Version for Edge Hostname(s)

● IPv4 only
Support only IPv4 traffic

○ IPv4 + IPv6 (dual stack)
Support both IPv4 and IPv6 traffic

IP version selection

4. If your site supports IPv6, select **IPv4 + IPv6**. If not (or if you are not sure), select **IPv4 only**. Click **Next**.

   The Review Edge Hostnames window appears. The sharding edge hostnames you entered are displayed in the Property Hostname column.

5. Select a certificate. For information about certificates, see the Property Manager online help.
6. Set the value in the Edge Hostname column so that it matches the value in the Property Hostnames column. You will do this for each sharding edge hostname individually. The process is slightly different for HTTP-only and HTTPS properties.

If…	Then…
HTTP-only properties	• Click the edit button (✎) for the first entry. A window appears prompting you to create or select an edge hostname. • Select **Create Edge Hostname**, click the input field, and enter the first portions of the sharding edge hostname. *Omit* "`edgesuite.net`". • On the drop-down menu to the right of that input box, select `edgesuite.net`. • Click **Update**. The view returns tot he property hostnames wizard. • Repeat for each sharding edge hostname.
HTTP/HTTPS properties	• Click the edit button (✎) for the first entry. A window appears prompting you to create or select an edge hostname. • Select **Custom**, click the **Custom CNAME Target field**, and enter the full sharding edge hostname. *Include* "`edgekey.net`". • Click **Update**. The view returns tot he property hostnames wizard. • Repeat for each sharding edge hostname.

7. Confirm that the property hostname and edge hostname shown for each shard are the same and that they match the sharding edge hostnames you noted earlier.
8. Click **Submit**.
   A success message appears, which you can dismiss. The view returns to the Property Manager Editor.

Next steps

To continue setting up FEO, configure the rules for the FEO resources property.

Configuring Rules

In this section, you will use the FEO NetStorage group you created earlier to set NetStorage as the origin for your FEO-optimized content. Also, you will set various behaviors to apply to FEO-optimized content.

This task applies to the FEO resources property configuration.

How to

1. Delete all rules except the Default Rule.
2. In the Default Rule (which has no match criteria), remove the **Allow POST** and **Real User Monitoring** behaviors.
3. In the Default Rule, modify the remaining behaviors as follows:

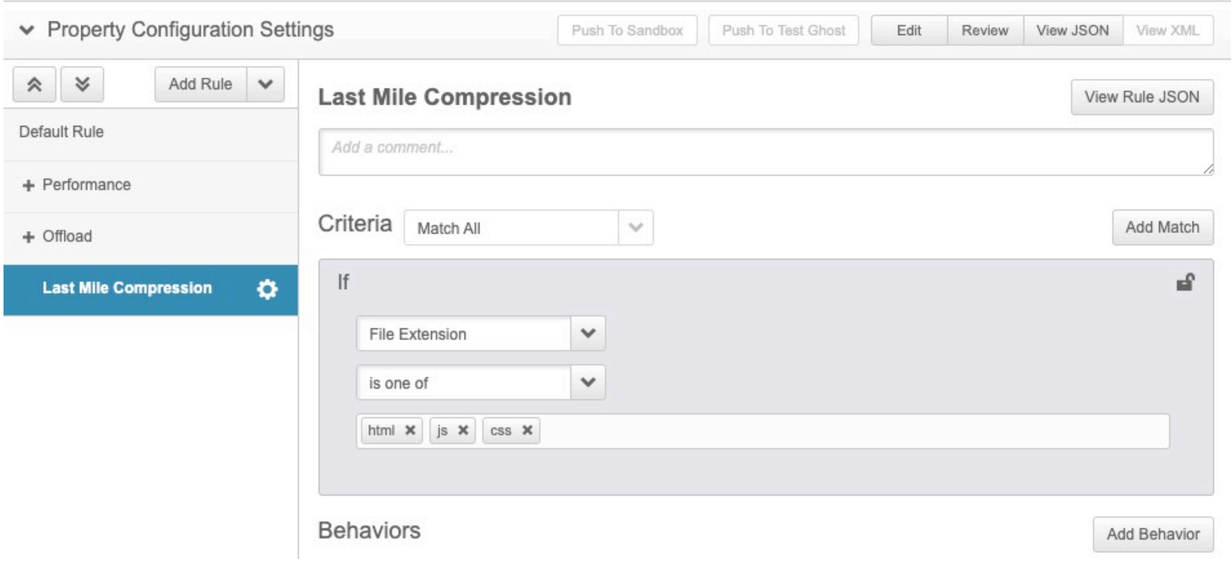
    a. **Origin Server:** Set **Origin Type** to NetStorage. Set **NetStorage Account** to the NetStorage upload account username create in *Setting Up NetStorage* on page 22.

    b. **Content Provider Code:** Use the NetStorage CP code create in *Setting Up NetStorage* on page 22.

4. In the Default Rule, add the following behaviors, configured as described in the example that follows these instructions.

    ⓘ **Note:** Every new version of an optimized resource has a unique filename. The long TTL (Max-age) configured here does not prevent modified resources from updating before the TTL expires.

5. Add a rule using the Blank Rule template, and name it “Last Mile Compression.”
6. In the Last Mile Compression rule, add a File Extension match criteria with the values html, js, and css.
7. In the Last Mile Compression rule, add the **Last Mile Acceleration** behavior and set **Compress Response** to **Always**.



Last Mile Compression

Criteria Match All

If

File Extension

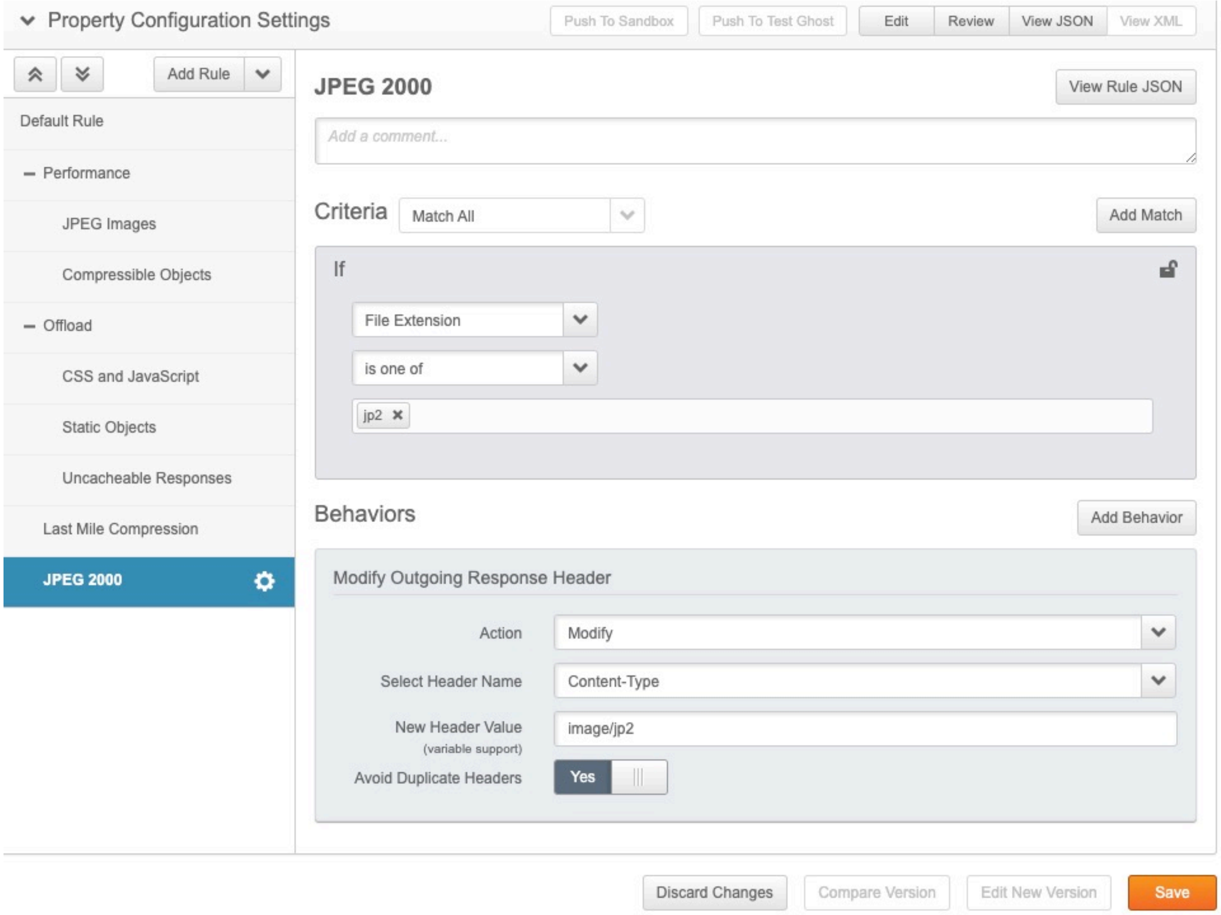
is one of

html ✖ js ✖ css ✖

Last Mile Compression rule in FEO resources configuration

8. Add another rule using the Blank Rule template, and name it “Unsupported Content-Types.”

9. Add a child rule to this rule, selecting the **Content-Type Override** template (located in the User Experience category).
10. Rename this rule “JPEG 2000,” set the **Match Criteria** to **If File Extension is one of jp2**, and configure the **Modify Outgoing Response Header Value** settings as follows:
    - Action: Modify
    - Select Header Name: Content-Type
    - New Header Value: image/jp2
    - Avoid Duplicate Headers: Yes



JPEG 2000

Criteria Match All

If

File Extension

is one of

jp2

Behaviors

Modify Outgoing Response Header

Action: Modify

Select Header Name: Content-Type

New Header Value (variable support): image/jp2

Avoid Duplicate Headers: Yes

JPEG 2000 rule in FEO resources configuration

11. Repeat Steps 9 and 10 to create a child rule named "JPEGXR" with these differences:
    - Match criteria: jxr file extension
    - New Header Value: image/vnd.ms-photo
12. Repeat Steps 9 and 10 to create a child rule named "WebP" with these differences:
    - Match criteria: webp file extension
    - New Header Value: image/webp

13. Confirm that there are no errors in the configuration, and then save it.

Example

This example shows how to configure the Default Rule for the FEO resources configuration.

Caching:

- Caching Options: Cache
- Force Revalidation of Stale Objects: Serve stale if unable to validate
- Max-age: 365 days

Prefect Objects:

- Enable: On

Tiered Distribution:

- Enable: On
- Tiered Distribution Map: Global (ch2)

Downstream Cacheability:

- Caching Option: Allow caching
- Cache Lifetime: Full edge TTL (max-age)
- Send Headers: Send only Cache-Control
- Mark as Private: Off

Cache HTTP Error Responses:

- Enable: On
- Max-age: 30 seconds
- Preserve Stale Objects: On

Cache Key Query Parameters:

- Behavior: Exclude all parameters

Enhanced Akamai Protocol: No settings required

Next steps

You have now completed all of the settings for the FEO resources configuration. Continue with the next task, activating this configuration.

Activating the FEO Resources Configuration

Once you have completed the FEO resources configuration, activate it.

Note: In order to test domain sharding functionality, you must activate your FEO resources configuration in production, not staging. If you do not have a live FEO configuration on production yet, this will not affect your existing parent product configuration. No end users are receiving pages that have been rewritten to include sharding edge hostnames.

How to

1. Navigate to the Property Details Page for your FEO resources configuration.
2. On the Actions menu (⚙▾) for your new FEO resources configuration (Version 1), select **Activate**.
   This displays the Property Manager Activate page.
3. Click the **Activate v1 on Production** button.
   The Production Network Activation window appears.
4. In the informational message, check the box to acknowledge.
5. Enter a comment in the **Notes** field that will identify this change. For example: “Activating FEO resources configuration on production.”
6. Confirm that the e-mail address where you wish to receive notifications appears in the **Address(es)** field. Optionally, enter additional addresses separated by commas.
7. Click the **Activate v1 on Production** button.

Example

Enter an example that illustrates the current task.

Next steps

Enter the tasks the user should do after finishing this task.

Using FEO's Configuration Manager

To set up and update your FEO configuration, you need to go to FEO's Configuration Manager, which is a configuration application accessed from the Akamai Control Center. Although it has many similarities to Property Manager, it is a separate environment. Review these topics to learn how to access FEO Configuration Manager, navigate to its various pages, and use them to create and manage your FEO configuration.

In order to be able to view FEO Configuration Manager, both the account contract and the digital property you want to optimize must be provisioned to use FEO. Moreover, FEO should have already been selected as an optional feature of your parent product's configuration file (see *Enabling FEO* on page 21).

Assume your account is `example.com`, and for that domain you own the digital properties `example.com` and `example.net`. If you select the digital property `example.net` and it is *not* enabled for FEO, you will not be able to access the FEO Configuration Manager. However, if you select FEO-enabled `example.com`, you will.

Navigating to FEO's Configuration Manager

Follow these instructions to access FEO's Configuration Manager, where you can create and manage your FEO configuration.

Before you begin

Complete the initial steps in the setup process. This includes enabling FEO, setting up NetStorage to receive transformed resources, and activating a property to manage your FEO resources.

How to

1. Start on the Akamai Control Center home page (*https://control.akamai.com*)
2. Select the FEO-ready account you want to work with.
3. Select the property to work with.
4. In the Akamai Control Center UI, from the **Related apps** menu, select **Front-End Optimization**. This displays the FEO Configuration Manager History Page.

⚠ **Attention:**

If you do not see the Front-End Optimization link under the Related apps menu, there are two reasons this might be the case:

- You have selected your account, but not a property.
- You have selected a property that is not provisioned for FEO.

Next steps

Now that you have accessed FEO's Configuration Manager, learn the basics of navigating the application by familiarizing yourself with the top-level configuration pages.

Top-Level Configuration Pages

The FEO Configuration Manager consists of four main pages: Configuration History, Configuration Details, Create New Optimization Policy, and Edit Optimization Policy.

Synopsis of FEO configuration pages

The page	How to use it
Configuration History	• Review the high-level details of all your versions including creation dates, comments, and network statuses. • Control your FEO configuration on the production environment via one-click activation, deactivation, and content purging. • Manage your FEO configuration files by adding, cloning, comparing, and deleting versions.
Configuration Details	• Review policy elements for a configuration version, including associated match criteria and optimizations. • Manage which policies are active; delete individual policies entirely. • Test URLs to determine which policy would be applied. • Access the Create New Policy and Edit Policy pages. • Access the configuration's Global Exclusions and Resource Delivery Settings.
Create New Optimization Policy	From the Configuration Details page, click the Add Policy button to access the **Create New Optimization Policy** page. Once you are on this page, you can: • Assign a name to a new policy. • Build match criteria based on the URL of incoming requests and the requesting browser. • Optionally, set a URL constant to serve as the basis for all page transformations in the policy.

The page	How to use it
	• Select optimization methods, refine how the methods will be applied, and define content to exclude from the selected methods.
Edit Optimization Policy Page	• Modify generic components of the policy, including the policy name, URL matches, and browser matches. • Change the selected optimization methods, their settings, and any optimization-specific exclusions.

Configuration History

This page lists configuration versions with their current statuses on the staging and production networks. Use the FEO Configuration History page to manage all facets of FEO Configuration Versions.

The Configuration History page is the home page for FEO's Configuration Manager. If you are on another page in the application, you can return to the Configuration History page. Save any changes on your current page, which returns you to the previous page. Continue to save changes as you work your way back to the Configuration History page. (The name of the button that saves changes differs depending on the page you are on.) If you don't care to save your changes (which is not advised), you can click the Cancel buttons to return to the topmost page.

The controls on the Configuration History page allow users to:

- monitor the history and statuses of configuration versions
- clone, edit, delete, and compare configuration versions
- purge transformed HTML from both staging and production
- quickly deactivate all FEO configurations with a single click
- activate/deactivate configuration versions on staging and production

Which capabilities are available at any given time vary depending on the current state of FEO, and the status of existing FEO configuration versions. For example, if a configuration version is active, it can be cloned but not edited or deleted. A configuration that has never been active can be edited, deleted, or activated. Additionally, some activities, such as deactivating and purging, can be done in several different ways.

Controls for the FEO Configuration History Page

This reference describes the indicators and controls on the Configuration History page. The availability of some controls depends on a configuration version's status.

One key element of the Configuration History page is the production status indicator. This shows the current state of FEO in the production environment. It does not reflect FEO status in the staging (or any other) environment.

FEO production status indicators

Indicator	What it means
FEO is Active	Signals that FEO is active on the production environment. There is an active FEO configuration file on production. That version is marked in the history table with a green bullet.
FEO is Pending	Indicates that there is an FEO configuration awaiting either activation or deactivation on the production network. In the history table, the pending version is marked with an orange bullet and a text label.
FEO is Inactive	Indicates that there are no active FEO configurations on production. The most recently active configuration is marked in the history list with a red bullet.
FEO is Disabled	Indicates that FEO is not currently active in production. This is because FEO is not enabled in the parent product configuration, there is no FEO configuration active in production, or both.

Configuration History page controls

Item	Description	Action availability by version status		
		New	Active	Inactive
Activate button	Appears only when FEO is inactive. Click to activate the most recently active configuration on production. To activate a different configuration, or activate a configuration on the staging environment, use the Action tool.			Yes
Deactivate button	When FEO is active, the Deactivate button appears. Use this button to remove the currently active configuration from the production environment—without pushing a different version in its place. Immediately after selecting Deactivate, FEO moves to the pending state, which means you cannot activate any other versions. Selecting Deactivate prevents the renewal of your FEO configuration after your parent product's TTL has expired. Once this happens, FEO will revert to delivering the unoptimized content from your origin. Deactivating a production configuration purges optimized content.		Yes	

Item	Description	Action availability by version status		
Create New Version button	Click to display a new-version page with no prefilled settings. When you save the new version, it appears in the history table. Until then, it has no version number and is marked as a draft.	Yes	Yes	Yes
Edit Version menu option	Available only for new configuration versions, which appear in the history grid with only a version number and creation date. To edit a specific version, use its corresponding Action tool.	Yes		
Clone Version menu option	Use the clone action to edit an existing configuration. The cloned version will be assigned a new version number and will remain editable until it is first activated on a network.	Yes	Yes	Yes
Delete Version menu option	New configuration versions can be deleted. Additionally, previously active configurations that were active on the staging network but never on production can be deleted. ⓘ **Note:** An inactive version can be deleted, but only if the version has never been active on any environment.	Yes		Yes
Activate Version menu option	FEO uses the metadata contained in the activated configuration to determine how to apply FEO optimizations to the associated property's content. Use this option to activate the existing configuration version of your on the network you specify. There can be only one active configuration version for each network (staging and production) at any given time.	Yes	Yes	Yes
Deactivate Version menu option	The result of deactivating a configuration is that FEO will not use the configuration's metadata as a basis for transforming content. Deactivating a configuration means that another configuration can be activated instead.		Yes	
View Metadata menu option	Selecting View Metadata opens a new browser tab that displays the configuration's underlying metadata.	Yes	Yes	Yes

Item	Description	Action availability by version status		
	This is the metadata auto-generated by the FEO Configuration Manager.			
Purge button and Purge Transformations menu option	Purging transformations removes FEO-optimized content from edge-server caches. It is possible to purge content from both the production and staging servers, but each environment must be purged separately. Once transformed content is purged, the content that Akamai subsequently delivers to the user depends on the status of your FEO configuration and the state of FEO as a feature of the parent product.		Yes	
Compare Versions button	Check any two configuration versions and click Compare Versions. This opens a window highlighting the configuration differences at the metadata level.	Yes	Yes	Yes

Enabling FEO

If the production status indicator reads **FEO is Disabled**, then FEO is not active in production. This is because at least one of the following is true:

- The FEO behavior in your parent product configuration is not enabled.
- There is no FEO configuration active in production.

You can activate an FEO configuration when FEO is disabled, but it will have no effect until you enable FEO in the parent product configuration. To enable FEO, go to your parent product configuration in Property Manager, set the Front-End Optimization behavior to **On**, and save the configuration.

Creating an FEO Configuration

Creating an FEO Configuration is simple. Do the following:

How to

1. Start from the Front-End Optimization Configuration History page.
2. Click the **Create New Version** button.

   Note: You can have an entirely empty FEO Configuration.

   Creating an FEO Configuration with defined settings and options is more involved. You need to add and define at least one optimization policy. For your first FEO Configuration, one policy is a good place to start.

FEO does not assign a version number to your new configuration version until you save it. Until then, the Configuration Details page indicates that it is a draft.

Adding an Optimization Policy

You can add a policy to a new configuration version, to a configuration version you are editing, or a clone of an existing configuration version. You select the configuration version from the FEO Configuration History page and then go to that version's Configuration Details page:

- click the **Create New Version** button, or
- select an existing version’s action button and then select **Edit Version**, or
- select an existing version’s action button and then select **Clone Version**.

This displays the *Configuration Details page*.

Front-End Optimization Configuration Details

The main page for managing an FEO configuration version is the Details Page. This is where you can get the best snapshot of your existing policies, their composition, and their status. Use the Configuration Details page to:

- review policy elements for a configuration version, including associated match criteria and optimizations,
- manage which policies are active, or delete individual policies entirely,
- test URLs to determine which policy would be applied,
- access the Create New Policy and Edit Policy pages,
- access FEO’s Suggested-Policy Tool, and
- access the configuration's Global Exclusions and Resource Delivery Settings.

Adding a New Policy to Your Configuration

To create and define a new policy, select **Add Policy**.

This displays the *Create New Optimization Policy* page where you can build out a policy based on whatever combination of available optimization methods, settings, and exclusions you like.

Changing an Existing Policy

How to

1. Select the action button for the policy you want to change.
2. Select **Edit Policy**, **Clone Policy**, or **Delete Policy**.

Testing Optimization Policy URL Matches

URL matches making extensive use of wildcards and regular expressions can seem inscrutable. So you may want to be sure the URL matches you define actually correspond to the pages you expect. Conversely, you may want to ensure specific web pages will be optimized and confirm which optimization methods will then apply. You can do this as part of managing your policies on the Optimization Policy tab of the FEO Configuration Details page.

FEO allows you to input a specific URL to test against the existing URL matches across all of your policies. Doing this will show you which policy will match and apply to the URL. Keep in mind that a single URL will only match one policy.

Enter a test URL to determine which policy it matches. FEO will flag the matching policy, whose optimization methods would be applied to the test URL if it comes in as a request. If no policies match, or the results are not what you expect, you may need to modify one or more URL matches.

URL tests consider only the URL matches of the FEO configuration's policies. Browser matches—which allow you apply policies (or not) depending on the requesting browser—are not relevant in this context.

ⓘ **Note:** Before testing a URL, save any changes you have made to your FEO configuration by using the Save Configuration button on the Configuration Details page. Changes include adding, modifying, reordering, and deleting the configuration's policies.

How to

1. At the top of the Configuration Details page , select the **Test URL Match** button. This displays a new panel: **TEST URL-TO-POLICY MATCHES**.
2. In the URL text field, enter the exact URL to test.
3. Select the **Test** button.
4. Look below in the **Manage Policies** panel. If a policy matches, it will have a green checkmark next to it.

   ⓘ **Note:** Although you will need to assign optimization methods and at least one selected browser to your new policy for it to provide any benefit, you are not required to do so. You will still be able to test the policy's URL matches against your property's URLs. Consider delaying the selection of optimization methods until you are satisfied that your URL matches work as intended.

Configuring Resource Delivery

The settings for NetStorage, sharding, and virtual paths are configured on the Resource Delivery tab. In order to understand these settings, you should be familiar with the concepts covered in *Considerations for Domain Sharding* on page 18.

Each pane on the Advanced Settings tab addresses a different question:

1. Where will FEO-optimized resources be stored?
2. What method of resource delivery do you want to use for HTTP requests?
3. What method of resource delivery do you want to use for HTTPS requests?

ⓘ **Note:** Before you can save your FEO configuration, you must enable and configure HTTP or HTTPS Resource Delivery (or both). If both types of resource delivery are disabled, the configuration will not save.

Prerequisites: FEO NetStorage Group and Resources Configuration

Configuration of NetStorage settings is described in *Starting the FEO Setup Process*. Once you complete those steps, you will have created your FEO configuration, but it won't have any match criteria, exclusions, or policies yet.

ⓘ **Note:** NetStorage settings are required for FEO to work.

In addition, your sharding domains should also be configured in your FEO resources configuration before you continue with FEO configuration. For more information, see:

- *Considerations for Domain Sharding*
- *Creating and Activating the FEO Resources Property* on page 24

ⓘ **Note:** FEO will not work unless resource delivery is enabled and configured for HTTP or HTTPS (or both).

Configuring Resource Delivery

To configure the resource delivery settings for HTTP and HTTPS requests, refer the following:

- *FEO and Domain Sharding*
- *Configuring Domain Sharding for HTTP Requests*
- *Configuring Resource Delivery for HTTPS Requests*

FEO and Domain Sharding

This is a brief explanation of domain sharding and how it relates to FEO. See also:

- *Considerations for Domain Sharding*
- *Configuring Domain Sharding for HTTP Requests*
- *Configuring Resource Delivery for HTTPS Requests*

A Brief Introduction to Domain Sharding

When you select the resource delivery method, you will decide whether to create domain shards. This is a technique for working around a common limitation of web browsers.

Here is the challenge: Often, browsers put a limit on the number of maximum simultaneous connections to each domain. Usually, a page will have a much larger number of resources than there are connections, so a queue begins to form.

For example, a browser that allows two connections will request two resources. No additional resources can be requested until one of the first two items finishes loading and a connection becomes available. This can slow load times, especially for pages that contain many external resources.

Domain sharding provides a solution to this problem by creating more domains so the browser opens more connections. For example, if you create two domain shards, the browser opens twice as many connections. The initial queue is slightly shorter, but more importantly, there are now twice as many connections. A browser that allows two connections per domain will load four resources simultaneously instead of two, so it will load the queued resources faster.

You do not need to maintain separate origin servers for each domain. Instead, you create CNAME entries that resolve each domain shard to an edge hostname dedicated to FEO resources.

How FEO Handles Domain Sharding

To distribute the requests for resources across the domain shards, FEO rewrites the URLs for those resources. URLs that look like this:

```
<image src="http://www.mydomain.com/images/logo.jpg">
<link rel="stylesheet" ref="http://www.mydomain.com/resources/product.css">
```

...might be rewritten like this:

```
<image src="http://feo1.mydomain.com/9CK2LD93KC9CV0S97.jpg">
<link rel="stylesheet" ref="http://feo2.mydomain.com/DJ934890SCAF8732.css">
```

FEO will distribute the different shard numbers among the URLs so that approximately equal numbers of resources will be requested via each domain shard. When you configure domain sharding, you can specify the components of the domain name.

Also, FEO will replace the filename of the origin resource with the FEO-generated filename of the optimized resource. These identifiers are unique for each version of a file, so longer caching periods (max-age) can apply to optimized resources.

When you configure domain sharding, you will indicate how many domain shards FEO should use and how FEO should rewrite the URLs.

Configuring Domain Sharding for HTTP Requests

This section explains how to configure domain sharding for HTTP requests. For instructions about HTTPS requests, see *Configuring Resource Delivery for HTTPS Requests*.

See also:

- *FEO and Domain Sharding*
- *Considerations for Domain Sharding*

1. Navigate to the FEO Configuration Details page.
2. Find the FEO configuration you created earlier when you set up your FEO NetStorage group. On its Actions menu, select **Edit Version**.

3. Select the **Resource Delivery** tab. The NetStorage settings that you generate earlier should still be there.

4. In **Configure Resource Delivery for HTTP Requests**:

   a. Set **HTTP Resource Delivery** to **Enabled**. This expands the HTTP resource delivery settings pane.

   b. Set **Number of Shards** to 2.

   c. On the **Look up Hostnames for Sharding Domains** menu, select either of the sharding edge hostnames you created in your *FEO resources configuration*. The Build Sharding Domains fields auto-populate. Refer to the figure below to see how these fields map to the elements of rewritten URLs.

5. Click **Show Domains**. The sharding domains appear, similar to those below. They should match the sharding edge hostnames you created.

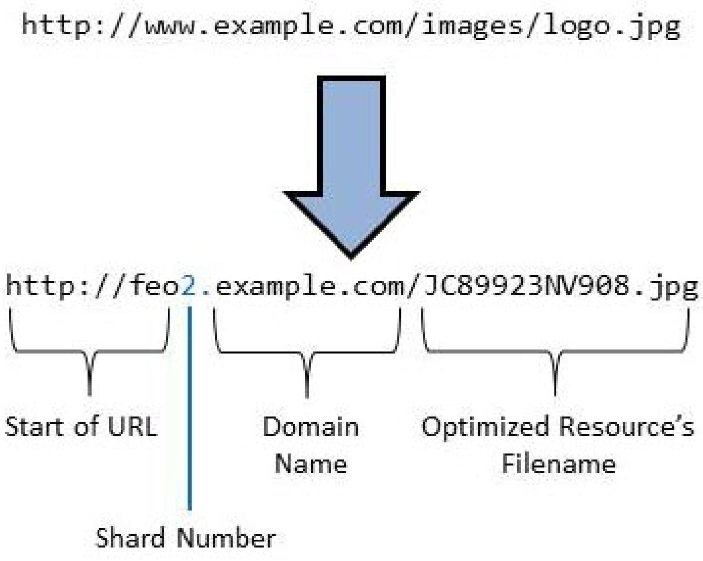
   - `http://feo1.example.com.edgesuite.net`

   - `http://feo2.example.com.edgesuite.net`

6. Click the **Validate** button (just below and to the right of the sharding domains). FEO attempts to write a test object to your FEO NetStorage group and to read it. If you get a failure message, contact customer support for assistance.

7. Click **Save Configuration**.

ⓘ **Note:** The shard number (<Shard #>) represents a *variable*. The Number of Shards field indicates the maximum value of that variable. For example, if you set Number of Shards to 3 and the Start of URL to http://feo, FEO would use URLs beginning http://feo1, http://feo2, and http://feo3. The *value* of the variable is indicated in blue in the figure.


http://www.example.com/images/logo.jpg
http://feo2.example.com/JC89923NV908.jpg
Start of URL
Domain Name
Optimized Resource's Filename
Shard Number

Example of URL Rewrite

Configuring Resource Delivery for HTTPS Requests

This explains how to determine what method of resource delivery to use for HTTPS requests and how to configure those methods. For instructions about HTTP requests, see *Configuring Domain Sharding for HTTP Requests*.

See also:

- *FEO and Domain Sharding*
- *Considerations for Domain Sharding*

How to Configure Resource Delivery for HTTPS Requests

As explained in *Considerations for Domain Sharding*, it is often better not to use domain sharding when delivering resources for HTTPS requests. If you wish to do so, enable HTTPS resource delivery, select **Use domain sharding for HTTPS requests**, and *follow the same instructions as for HTTP*. There are a few key differences:

- **Choose sharding domains that are on your own domain**, like `feo1.example.com` and `feo2.example.com` . It is not possible to use edgekey.net domain shards for HTTPS requests.
- **Install a certificate on your sharding domains.** Performance is usually best if you use the same certificate as that used for your main domain (like `www.example.com`). See the Certificate Provisioning Service User Guide for instructions. Because you will be assigning multiple domains to a single certificate, you must use either a wildcard or a SAN certificate.
- **Use your sharding domains to deliver FEO resources.** When creating your FEO resources configuration, create Property Hostnames entries that map each of your custom sharding domains to itself.
- **CNAME to your sharding domains.** Just as you would for your parent product configuration, create the DNS CNAME record with your registrar that CNAMEs each sharding domain to your resources domain.

But it is usually better to bypass domain sharding for HTTPS requests by configuring a virtual path.

To configure a virtual path:

How to

1. Navigate to the FEO Configuration Details page and select the **Resource Delivery** tab.
2. In the third pane (**Configure Resource Delivery for HTTPS Requests**), enable HTTPS Resource Delivery. The resource delivery settings appear.
3. Select **Use virtual path for HTTPS requests**.The virtual path settings appear.
4. Enter a value for the virtual path. For example, `/feo-cdn` .
5. Save this FEO Configuration.
6. In Property Manager, create a new version of the *parent product configuration*.
7. Add a new rule using the Blank Rule template. This rule will handle requests that contain the FEO Virtual Path.
8. Set the match criteria for this rule to find requests where the path matches your FEO virtual path. For example: Path matches one of `/feo-cdn/*`

9. Add the behaviors in the table *Parent Product Configuration for Virtual Path* to this rule.
10. Add a rule using the Blank Rule template. This rule will exclude certain content that NetStorage does not support.**Note:** Support for these formats is currently in beta. If you are participating in this beta, skip this step.

    a. Add a child rule using the Content-Type Override template (in the User Experience category).

    b. On this child rule's Action menu, select Duplicate. Do this twice so you have three instances of the rule.

    c. Configure these rules as shown in the table *Parent Product Configuration for Unsupported Content Types*.
11. Save your parent product configuration.

Parent Product Configuration for Virtual Path

Behavior	Field	Setting
Content Provider Code	Content Provider Code:	Select the CP Code you have designated for FEO.
Origin Server	Origin Type: NetStorage Account:	NetStorage The account you are using for FEO-optimized resources
Caching	Caching Option: Force Revalidation of Stale Objects: Max-age:	Cache Serve stale if unable to validate 365 days
Last Mile Acceleration (Gzip)	Compress Response:	Always
Modify Outgoing Request Path	Action: Find what: Occurrences: Keep the query parameters:	Remove part of the incoming path Your FEO virtual path (for example, /feo-cdn/) First occurrence only No
Cache HTTP Error Responses	Enable: Max-age: Preserve Stale Objects:	On 30 seconds Off
Cache Key Query Parameters	Behavior:	Exclude all parameters
Downstream Cacheability	Caching Option: Cache Lifetime:	Allow caching Full edge TTL (max-age)

Behavior	Field	Setting
	Send Headers: Mark as Private:	Send only Cache-Control Off

Parent Product Configuration for Unsupported Content Types

Rename Rule to...	...and Configure These Settings
JPEG 2000	Match criteria: Set the list of the file extensions to include only jp2 Action: Modify Select Header Name: Content-Type New Header Value: image/jp2 Avoid Duplicate Headers: Yes
JPEGXR	Match criteria: Set the list of the file extensions to include only jxr Action: Modify Select Header Name: Content-Type New Header Value: image/vnd.ms-photo Avoid Duplicate Headers: Yes
WebP	Match criteria: Set the list of the file extensions to include only webp Action: Modify Select Header Name: Content-Type New Header Value: image/webp Avoid Duplicate Headers: Yes

The Create New Optimization Policy Page

The Create New Optimization Policy page allows you to create a new policy from scratch. To access this page, select the **Add Policy** button on the *FEO Configuration Details page*.

Adding Policies

For those new to policy creation, we recommend that you include all available browsers and three key optimizations: Optimize CSS, Optimize Images, and Optimize JavaScript. Add these to your policy and enable all of each method's optional settings as well. These core optimizations are the most stable across browser and content types. Nevertheless, always test new policies in the Staging environment before deploying to Production.

Before you begin

Choose the right combination of optimizations for your new policy. Before crafting a new policy, consider reviewing *Optimization Methods* on page 47, which covers each optimization in detail with definitions, examples, best practices, cautions, and help with exclusions.

For complete field descriptions, see *Controls for the Create New Optimization Policy Page* on page 45.

How to

1. Start from the FEO Configuration Details page.
2. Select **Add Policy** to display the Create New Optimization Policy page.
3. Enter the policy name, which you may change at any time without affecting the way the policy works.
4. Complete the Match Criteria:
    - Add one or more URL matches to indicate to which pages FEO will apply this policy
    - Select one or more browsers. FEO will apply this policy to requests from the selected browsers only.
5. So that FEO always builds the policy's analysis rules based on the same page, enter a URL Constant. Leave this field empty to build analysis rules based on the first-requested page.
6. Select the optimization methods to use for the policy by clicking on the methods' plus-sign buttons. Once selected, the methods moves from the Available to the Selected Method columns.
7. Configure optional settings for your selected optimizations. To do this, click on each optimization method listed in the selected column to bring up its optional settings.
8. If there are resources on your site that meet the policy's URL matches but should not be transformed by all of the policy's optimizations, use exclusions to protect these resources. For each method that should not apply to all matching resources, select the method's Exclusions tab to add exclusions.
9. When you are done, select **Save Policy** to return to the Configuration Details page, where you should select **Save Configuration** to ensure your new policy is preserved.

Next steps

Although you will need to assign optimization methods to your new policy for it to provide any benefit, you are not required to do so. You will still be able to test the policy's URL matches against your property's URLs. Consider delaying the selection of optimization methods until you are satisfied that your URL matches work as intended. See *Testing Optimization Policy URL Matches* on page 38 for more information.

Controls for the Create New Optimization Policy Page

Refer to this table to understand how to use the controls on the Create New Optimization Policy page.

Controls for the Create New Optimization Policy page

Control	Description
Policy Name	A unique identifier that indicates the type(s) of pages to which this policy will apply, for example *Product Descriptions or Checkout Pages*. Each policy must have a unique name, but any format will work.

Control	Description
Match Criteria	There are two types of match criteria: URL matches and browser matches. Use one or more *URL matches* (either wildcard or regular expression) to indicate the pages to which this policy applies. Note that URL matches are OR matches, so a page that matches only one of the criteria is still a match. ⓘ **Note:** Do not use wildcards in the hostname portion of the URL, only in the path and filename. For example, `http://*example.com/` is not valid. Use *browser matches* to create distinct policies for different browsers.
Use URL Constant as Policy Benchmark	By default, FEO's Analyzer creates a set of transformation instructions (or bundle) based on the resources of the first-requested URL that maps to a policy's URL matches. This means that FEO will optimize the resources for all subsequent URLs in this policy the same way as for the first-requested page. You have no control over which is the first-requested page, however. If your policy's URL matches are broad, this could lead to inconsistent transformations. Should you desire more consistency and control, enter a URL Constant. The FEO Analyzer will then use the URL Constant (not the first-requested URL) as the basis for transformation instructions.
Optimization Methods	Use this panel to add optimizations to your policy, manage settings, and configure local exclusions.

The Edit Optimization Policy Page

Use the Edit Optimize Policy Page to change a policy's name, match criteria, URL constant, optimization methods, settings, and exclusions. It looks the same as the Create New Optimization Policy page.

How to

1. Start from the Configurations Details screen and find the policy you want to edit.
2. From the that policy's Actions menu, select the Edit Policy option.
3. Make your desired edits and select **Save Policy**.
4. When you return to the Configuration Details page, be sure to select **Save Configuration**.
5. Continue to re-test the policy to confirm your edits work as expected.

Next steps

Before leaving the page or closing the browser window, save the policy and then save the configuration. Otherwise, you may lose your edits. After you have completed saving your work, re-test the policy in the Staging environment.

Optimization Methods

FEO makes available optimization methods to enhance the performance of your page. Methods vary by behavior, complexity and configurability, and resource-type applicability. Additionally, some methods only apply to content served over mobile networks.

ⓘ **Note:** Customers with FEO Standard have access to a limited set of optimization methods. Review the table for a list of features that are provided with FEO Standard.

Feature	Available with FEO Standard	Notes
Asynchronous JavaScript	Yes	
Cellular Connection Keep-Alive	No	This feature will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations. 📱 This feature is only useful in instances of HTTP requests sent over cellular networks.
Critical CSS for Mobile	Yes	This feature will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations. 📱 This feature is only useful in instances of HTTP requests sent over cellular networks.
Defer Print Style Sheets	No	
DNS Prefetching	Yes	
EdgeStart	Yes	
Extend Resource Cache	No	
Invoke Click On-Touch	No	This feature will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations.

Feature	Available with FEO Standard	Notes
		This feature is only useful in instances of HTTP requests sent over cellular networks.
JavaScript Pre-Execution	No	This feature will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations. This feature is only useful in instances of HTTP requests sent over cellular networks.
On-Demand Image Loading	Yes	
Optimize CSS	Yes	
Optimize Images	Yes	
Optimize JavaScript	Yes	
Page Prefetching	No	This feature will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations.
Resource Prefetching	No	

Asynchronous JavaScript

In some cases web pages containing JavaScript do not render HTML content until after all JavaScript processing completes; the JavaScript blocks all other activity. Enabling FEO’s Asynchronous JavaScript optimization method overrides this browser behavior, allowing all HTML and CSS content to render prior to any JavaScript being executed.

As a result, users do not have to wait for a page’s scripts to complete before seeing page content. For pages that are JavaScript intensive, this optimization method can dramatically reduce rendering time. Asynchronous JavaScript is particularly useful for pages with calls to third-party scripts because delayed scripts from another domain will not keep your page from rendering.

FEO’s Asynchronous JavaScript behavior is compatible with all recent browsers. These include the following:

- IE 8.0 and above
- Chrome

- Firefox
- Opera
- Safari
- Safari on iOS (iPhone, iPad, iPod)
- The stock Android browser

Optional Settings

Use zero or more additional settings to further refine how to implement Asynchronous JavaScript for the current policy.

Asynchronous JavaScript Optional Settings

Setting	Description
Avoid Global ID Collisions	Ensures that global JavaScript variables are defined before any element IDs with the same name.
Conditionally Defer Execution	Executes JavaScript based on the existence and location of calls to external scripts that occur in the requested URL's HTML file. With this setting enabled, FEO executes inline JavaScript before onload provided that the code comes before calls to external scripts. Upon the first encounter of a <script> tag that contains a "src" attribute, FEO defers executing all remaining JavaScript (both internal and external) until after the onload event completes. **Conditionally Defer Execution** and **Defer Execution Until Onload** are mutually exclusive. You cannot select both.
Defer Execution Until Onload Event	Prevents JavaScript from executing until after the on-load event fires. **Conditionally Defer Execution** and **Defer Execution Until Onload** are mutually exclusive. You cannot select both.
Enable HTML5 Elements in IE	Supports HTML5 elements for older browsers that would not otherwise allow this.
Responsive Resource Load Ordering	Reorders JavaScript and CSS calls for faster page rendering when the requesting client is a mobile device. Delays download and execution of JavaScript until after the on-load event. Typically, Responsive Load Ordering delays JavaScript execution but does not delay downloads.

Setting	Description
	In order to implement Responsive Resource Load Ordering, FEO requires selection of the HTML5 Advanced Cache setting for Optimize JavaScript.

Exclusions

For this optimization, you can use either a blacklist (exclusion) or a whitelist (inclusion) approach.

The blacklist approach is similar to other optimizations. First select which kind of JavaScript you want to protect: Exclude External JavaScript Files or Exclude Inline JavaScript. Any excluded JavaScript will be executed by the browser during page load, which is consistent with the browser's default behavior. Assume that inline JavaScript is contained within the HTML page that the browser has requested. External JavaScript is any code in its own .js file. This would include third-party scripts outside of your own domain.

To exclude an external JavaScript file, enter an exclusion pattern that matches on any part of the JavaScript URL. To exclude inline JavaScript, enter an exclusion pattern that matches on the content of the script. Use regular expressions.

ⓘ **Note:** You can apply either exclusions or inclusions to external scripts, not both. However, you can use inclusions for external scripts with exclusions for inline scripts.

The whitelist approach creates an inclusion rather than an exclusion. To use this approach, select Include JS and exclude everything, and then use a regular expression to identify the only JavaScripts that should load asynchronously. All other scripts will be excluded.

Whitelisting is only available for external JavaScript files, not inline JS. To include an external JavaScript file, enter an inclusion pattern that matches on any part of the JavaScript URL. Use regular expressions.

Example Scenarios for Excluding JavaScript

Scenario	Exclusion Pattern
If you use inline JavaScript to determine the user's browser type, and the browser type affects which style sheet to use, then you need to execute this inline code before the page renders. Enter a regular expression to exclude the function `getBrowserType()` from Asynchronous JavaScript	`function\getBrowserType\(\)`
If you use a separate JavaScript file to determine browser type, you need to exclude the entire script.	`getBrowserType\.js`

ⓘ **Note:** When implementing Asynchronous JavaScript, FEO maintains script execution order. For this reason, you need to account for script dependencies when you are creating exclusions. For example, if you exclude the script `foo.js` , which is dependent on `bar.js` , you should exclude `bar.js` as well. Failure to do so can lead to JavaScript errors. In the case of Asynchronous JavaScript, this will cause the dependent script to execute `after` the excluded script, instead of before.

Cautions

For pages that generate essential visual content using JavaScript, deferring execution of JavaScript may not lead to a user-perceived increase of page speed. In some such cases, you will want to exclude these sources from this optimization method.

To prevent content-rendering problems, FEO examines the requesting-device's user agent and disables Asynchronous JavaScript if the user's browser is non-compliant. You need not accommodate this possibility.

Best Practices

Because FEO dynamically disables Asynchronous JavaScript for browsers that cannot accommodate it, there is minimal risk with using this optimization method. We recommend adding it to all optimization policies by default. Use this method for pages that contain enough static content to occupy visitors while JavaScript downloads and executes in the background.

Before going live, activate on the Akamai staging network and test thoroughly. Make certain the content looks as expected, particularly with older browsers and pages with a lot of JavaScript-generating content. If pages prove to be incompatible with this method, use exclusions to shield just those pages.

Cellular Connection Keep-Alive

Only for content served over mobile networks.

Important: This FEO feature will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations.

Every time a mobile-based browser requests a web page, the cost of establishing a new connection to the closest tower can introduce a two- to three-second delay. FEO can eliminate the cost of recurrently renewing connections. Here is how: the Cellular Connection Keep-Alive method maintains an open connection by sending dummy-page requests when genuine HTTP requests are unavailable. Reducing the number, and start-up time of new connections can reduce latency. This method applies to iPhone, iPad and Android devices.

Cautions

This optimization method may increase the resources your device needs to access your content.

Best Practices

Consider using a URL match that casts a large net. FEO will simply ignore any matches on non-mobile content when applying this optimization method.

Optional Settings

None.

Exclusions

None.

Critical CSS for Mobile

Only for content served over mobile networks.

Important: This FEO feature will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations.

When a page begins loading, it's best to get the visible part of the page loaded as quickly as possible. One strategy for speeding page load times is to defer below-the-fold content and resources until after the first content has loaded. Critical CSS for Mobile applies this strategy to external style sheets.

Critical CSS for Mobile identifies which specific CSS style elements from external style sheets are used above the fold. It retrieves these styles and embeds them in the optimized page. This shortens the time to load the top part of the page. While the user views the above-the-fold content, the browser loads the full style sheets.

Critical CSS for Mobile applies to requests from mobile browsers only.

Note: The Critical CSS for Mobile optimization is incompatible with two optional settings: Edge CSS in EdgeStart and HTML5 Advanced Cache in Optimize CSS. A policy may only include one of these three.

Exclusions

For any style sheet you do not wish to optimize, enter a text-based pattern to match against the style sheet's URL. Or enter a pattern to exclude an entire directory of style sheets. Use regular expressions for exclusion patterns, which can match on any part of a URL for that URL to be excluded.

Examples of Exclusions for Critical CSS for Mobile

Scenario	Exclusion Pattern
You do not want to optimize a specific style sheet, for example **IE6OneOffs.css**.	`IE6OneOffs\.css`

Scenario	Exclusion Pattern
You want to leave alone all of the style sheets contained in one directory, for example **/ IEstyles**.	`/IEstyles/`

Defer Print Style Sheets

Use this method to delay the processing of style sheets for specific media types until after the page loads. For instance, it is usually unnecessary to process print-only styles until after the page displays. Media-type stylesheets you might want to defer could include print and projection. This optimization method can be helpful for pages that specify distinct style sheets for alternative media types.

By default, FEO will defer processing the print style sheet. However, you may change this at the time you add Defer Print Style Sheets to an optimization policy. Enter media types as comma delimited strings. Use the literal media-type names, not regular expressions. Valid entries include the following:

- aural
- braille
- embossed
- handheld
- print
- projection
- screen
- tty
- tv

Scenario	String Match
Defer the *print* and *projection* style sheets.	`print, projection`
Defer the *tty* style sheet.	`Tty`

Cautions

If your site contains textually-dense pages such as magazine articles where the user's primary objective is to print, deferring the print style sheet may not be the optimal approach.

DNS Prefetching

This optimization method resolves and caches the DNS lookups for resources on a page before the linked content is requested. This approach is particularly effective for pages with many resources served from external domains, as well as those pages serving third-party content such as images and videos.

The functionality is slightly different for browsers that support preconnect. In these cases, the browser also establishes connections to the domains of linked resources. (As of October 2015, only the latest Chrome browsers support preconnect.)

Optional Settings

Pages using JavaScript to dynamically generate and render HTML may add resources to the dynamic content. Select the Enable pre-execution checkbox to resolve and cache the DNS lookups for dynamically-added resources before the JavaScript actually runs.

Exclusions

None.

EdgeStart

When EdgeStart is enabled, the edge server responds immediately to HTTP requests with a starter page containing static content. While the edge server retrieves the rest of the requested HTML from the origin server, the browser is already loading and displaying the resources specified in the starter page.

How It Works

The key component of EdgeStart is the starter page, which is a bare-bones file cached on the edge network; it includes the beginning part of the first-requested page's HTML. FEO uses the **top tag** as the cutoff point; this is usually the `<head>` tag, but if the `<meta>` tag contains certain attributes (described below), this becomes the top-tagged content. The content in the first-requested page that comes before the top tag becomes the starter content.

The following attributes in the `<meta>` tag qualify it as an EdgeStart top tag. If any of these are present, the `<meta>` tag will be the cutoff point, not `<head>` .

- `http-equiv="X-UA-Compatible"`
- `http-equiv="Content-Type"`
- `charset="{` *`anyValue`* `}"`

After the first-requested page's top content appears, the starter page calls an EdgeStart-generated script that will load static resources such as style sheets and scripts. (Yes, the scripts produce dynamic HTML, but the scripts themselves are static and can be loaded from the EdgeServer's cache.)

ⓘ **Note:** Remember that whichever page in the policy is first requested seeds the transformations for all of the policy's pages. The EdgeStart cache works the same way. This means that if there is any dynamically-produced HTML above the top tag, the resulting content is cached as static in the starter page and will appear for every page in the policy until the configuration expires.

Optional Settings

By default, the starter page includes only those CSS files in the beginning part of the analyzed page. Select Edge CSS to include *all* CSS files linked from the first-requested page in the starter page, even if they appear after the top tag.

ⓘ **Note:** The Edge CSS setting is incompatible with the Critical CSS for Mobile optimization and the HTML5 Advanced Cache optional setting in Optimize CSS. A policy may only include one of these three.

Exclusions

If Edge CSS is selected, you can indicate any style sheets you wish to exclude from this option. For any style sheet you do not wish to optimize, enter a text-based pattern to match against the style sheet's URL. Or enter a pattern to exclude an entire directory of style sheets. Use regular expressions for exclusion patterns, which can match on any part of a URL for that URL to be excluded.

Scenario	Exclusion Pattern
You do not want to optimize a specific style sheet, for example **IE6OneOffs.css**.	`IE6OneOffs\.css`
You want to leave alone all of the style sheets contained in one directory, for example **/IEstyles**.	`/IEstyles/`

ⓘ **Note:** The exclusions apply to the Edge CSS option *only*. They do not apply to any other EdgeStart resources.

To exclude any other content from the EdgeStart optimization, simply ensure that the content does not map to any policies with EdgeStart enabled. You would do this when setting up a policy's URL matches. There are no additional exclusions for EdgeStart.

Cautions

Take care that site pages do not have any dynamic content placed before the top tag in the origin page's HTML. For example, if your site uses dynamically-generated title tags across the site and the title tag is in the header, EdgeStart will render every page in the EdgeStart-enabled policy with the same title.

Best Practices

EdgeStart's performance gains are most evident under the following conditions:

- The origin server is geographically distant from the requesting browser.
- The origin is slow overall, or at least slow to deliver HTML.
- HTML comprises a significant percentage of the page load, such as tends to be the case for mobile sites.

EdgeStart is most effective when you have well-constructed policies in which pages are similar in structure and content, and when there is no dynamic content before the origin-HTML's top tag.

Extend Resource Cacheability

As a rule, FEO does not optimize resources with a short cache life. If a resource's Time to Live (TTL) in the cache is shorter than two hours, FEO will not apply any optimizations to it. The Extend Resource

Cacheability optimization provides an override mechanism. Adding this method to a policy forces FEO to optimize all cacheable resources, even those with very short TTLs.

Cautions

FEO optimizes copies of origin resources and serves these optimized resources from its own network. When the origin resources change, FEO optimizes and caches new versions of the resources. This process can take up to 15 minutes, which extends the TTL by the same amount of time. Optimizing resources with cache values that are less than 15 minutes means that FEO may return old-content versions to the browser. This possibility increases for base resources that change frequently.

Best Practices

Do not use this optimization method for resources that change more often than every 15 minutes, particularly if the most recent version must always be the version that Akamai serves. For example, a timer counting down to the end of a one-hour flash sale would be a poor candidate for extended caching.

Optional Settings

None.

Exclusions

To disqualify content from extended caching, exclude these resources from being optimized in the first place. To do this, go to the Method Settings tab of the appropriate optimization. For example, to exclude a .jsp file from extended caching, add the regular expression for the filename to the Optimize JavaScript's Exclusions tab. For style sheets, add the exclusion to the Optimization CSS exclusions. And so forth.

Invoke Click On-Touch

Only for content served over mobile networks.

Important: This FEO feature will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations.

By default, touch screen browsers fire an on-touch event 300 milliseconds before firing an on-click event, in order to allow for pinch, zoom, and other touch operations. Invoke Click On-Touch adds an on-touch event handler to hyperlinks, which eliminates the 300 millisecond navigation delay. This method applies to iPhone, iPad, and Android devices.

Cautions

Invoking hyperlinks on touch makes hyperlinks more sensitive to clicks. Scrolling and zooming on Click On-Touch enabled pages might require extra dexterity to avoid accidentally hitting hyperlinks.

Best Practices

Test this optimization method on a range of different page types using a variety of mobile devices. Test large pages with many hyperlinks and small pages with few. Try using mobile devices with smaller screens. Then weigh the on-click time savings against any usability issues so you may determine if, and when, to apply this optimization method.

Optional Settings

None.

Exclusions

None.

JavaScript Pre-Execution

Only for content served over mobile networks.

Important: This FEO feature will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations.

Executing JavaScript on mobile devices can be slow because mobile devices tend to have less-powerful processing capabilities than non-mobile devices. On average, mobile browsers take ten times longer than desktop browsers to process JavaScript. When JavaScript Pre-Execution is enabled, FEO executes much of the embedded JavaScript offline and provides the requesting browser with a mostly static page. For context-specific scripts requiring dynamic responsiveness, FEO loads all static content first before executing the remaining JavaScript.

Using this method reduces the amount of JavaScript that either the client or browser will need to execute at page-request time, lessening some of the mobile device's processing burden. Shifting the JavaScript execution to the browser, coupled with asynchronous implementation, reduces the number of HTTP requests and allows static content to load without delay.

JavaScript Pre-Execution is one of the FEO methods that analyzes a source page retrieved from the origin, rewrites the HTML, and sends back an optimized version of the page. Within this framework, this is how JavaScript Pre-Execution works

How to

1. FEO downloads the source-page's JavaScript and loads it into a headless browser. Think of the headless browser as a staging area where FEO can preview the JavaScript's output.
2. Once the page is completely loaded in the headless browser, and all JavaScript has run, FEO captures the output and extracts any generated text that is safe to serve as static content.
3. FEO creates an optimized version of the page, replacing JavaScript with the static content it would have produced. Any remaining dynamic-text producing JavaScript will be modified to run while the static content downloads. That is, the remaining JavaScript will execute asynchronously.

Best Practices

Design web pages so that there is enough static content to make the page usable before JavaScript executes. When determining exclusions for this optimization method, sketch out your desired combination of exclusion types and match criteria before entering them via the Configuration Manager. Read the **Pre-Execution Exclusions** discussion below for more information.

Optional Settings

This optimization method has no additional settings.

Exclusions

For JavaScript Pre-Execution Optimization, there are exclusion categories and, within those categories, exclusion criteria that you can apply.

Exclusion Categories

There are three categories of exclusions for the JavaScript Pre-Execution Optimization method:

- **Do Not Execute**This exclusion blocks execution of JavaScript if it corresponds to your defined match criteria. As a result, the excluded JavaScript will not execute until the browser requests the page, at which point the mobile device may require additional HTTP requests and processing overhead.

- **Do Not Extract**This exclusion executes the JavaScript and runs any of its functions, but does not inject extracted text into the optimized page. At the time of the page request, the browser will then still need to run all text-generating JavaScript. Use this exclusion to prevent the extraction of all text contained in one or more JavaScript files that correspond to your URL matches.

- **Dynamic Content Exclusions**This exclusion does not interfere with JavaScript execution or the extraction of text. Use Dynamic Content Exclusions to prevent FEO from injecting specific JavaScript-produced text (characters, words, phrases, etc.) into the optimized page. Use regular expressions to denote the dynamic content you want to exclude. Code that produces the excluded text will remain in the optimized page, and will generate the text dynamically at the time of the page request.

Exclusion Criteria

In addition to the three categories of exclusions, there are three criteria against which you can define your match criteria:

- **Name:** Build a regular expression that will match one or more JavaScript filenames (*.js). You can use the Name match criterion with the Do Not Execute and Do Not Extract exclusion types.

- **Code:** Build a regular expression that will match on code contained within one or more JavaScript files. Any JavaScript containing the defined code will then be excluded. You can use the Code match criterion with the Do Not Execute and Do Not Extract exclusion types.

- **Text:** Build a regular expression that will match specific text you do not want to inject as static content in the optimized page. This match criterion is only applicable for Dynamic Content Exclusions.

You build your JavaScript Pre-Execution exclusions on the Edit Optimization Policies page in the Configuration Manager as part of selecting the optimization methods for a specific policy. After you have added JavaScript Pre-Execution to your policy, use the **Method Settings** column to define your exclusions.

Adding Exclusions to a Policy's JavaScript Pre-Execution

Adding an exclusion requires three steps:

How to

1. Select the desired exclusion type/match criteria combination from the drop-down menu.
2. Click **Add Exclusion**.
3. Enter the regular expression against which the exclusion will apply. Note that you will not see a text field for your regular expression until after you enter your first exclusion and click the **Add Exclusion** button. You may define multiple exclusions with differing exclusion type/match criteria combinations.
4. Repeat steps 1 through 3 for any additional type/match criteria combinations you require.

What you should see

⚠ **Caution:** FEO only displays the match criteria for the currently-selected exclusion type. When you select a different exclusion type from the drop-down menu, FEO will hide all other match criteria. Any match conditions you have entered are not deleted, only hidden. To redisplay your other match criteria, reselect the appropriate exclusion type from the drop-down menu.

On-Demand Image Loading

Use this optimization method, sometimes referred to as Just-in-Time (JIT) image loading, to download and render images as they scroll into view. On-Demand Image Loading

- reduces image download size,
- helps pages render more quickly, and
- eliminates the performance costs incurred from downloading and rendering content that the end-user never sees.

Optional Setting: Low-Quality Image Preload

Important: The **Low-Quality Image Preload** setting will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations.

In addition to loading images just-in-time, you can also load images in stages so that rendering large images does not unnecessarily delay displaying the rest of the above-the-fold content. Instead of loading images after all other content, and leaving gaping holes on the user-viewable page, you can display fast-loading lower-quality image copies as placeholders.

ⓘ **Note:** The Optional Setting **low-quality image preload** can be used to reduce image resolution.

The Sequence of Image Rendering

Once the page has completely loaded, the browser will replace the stand-in images with their original counterparts. How long placeholders appear on the page will depend on the rest of the page's content.

Users may see the low-quality images only briefly. Alternatively, for heavyweight pages such as those with a lot of third-party content, users may see the placeholders longer.

Compressed Image File Formats

When FEO compresses images, it takes graphic files of any format (e.g., .jpeg, .png, .gif, etc.) and determines the smallest compatible format. So FEO may render an original .gif image as a .png for the placeholder copy if the .png is smaller. But if there is a transparent image, FEO will compress to a format that keeps the transparency, even if other formats might create smaller files.

Managing Compression Levels

You can control the degree of compression applied to the low-quality images by allowing or disallowing various compression techniques. To determine the best level of compression for your images, consider the placeholder images' display duration and the quality and detail viewers will need to distinguish the images' salient features.

Important: Akamai generates the low-quality images at the time of analysis while performing all other image optimizations. The low-quality images will have the same caching and purge behavior as any other image optimized by FEO. That is, FEO uses the property's Time-to-Live (TTL) settings to manage your placeholder images.

Optional Setting: Skip Hidden Images

If an image is not positioned on the screen when the page loads, you might not want to apply On-Demand Image Loading. For example, images contained in a collapsed panel won't be visible unless the user expands it. Enable this setting to exclude such images.

Optional Settings for On-Demand Image Loading

Setting	Description
Enable low-quality image preload	Checking this option signals that when retrieving the page content, FEO will need to • Find page's images on the origin server, • Create and cache compressed copies of the images, and • Serve the page's images in tiers, rendering the compressed image copies first and the original images only after the rest of the page finishes loading.

Setting	Description
	Important: The **low-quality image preload** setting will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations.
Allow indexed color	Indexed color uses a reduced-palette of 256 colors. Enabling this option sacrifices color purity for speed.
Set the desired chrominance quality	This option indicates the acceptable level of chroma resolution reduction for compressed JPEG images. The simplest way to think about chroma is the degree of saturation. So a low-chroma image will appear muted while a high-chroma image will appear vibrant. To reduce an image’s chrominance, select from the drop-down menu the degree of the reduction as either half or quarter.
Set the JPEG quality	Enter an integer between 1 and 100, with 1 representing the lowest possible image quality (and greatest compression), and 100 representing the opposite end of the spectrum. This quality level will only be used for placeholder images once FEO determines that a JPEG represents the smallest compatible file format.
Skip hidden images	Do not delay loading images that are not positioned when the page first loads.

Cautions

On-Demand Image Loading relies on JavaScript being enabled in the browser to dynamically load images as they come into view.

Best Practices

Should you choose to exclude some images from On-Demand Image Loading, be careful that your regular expression is not overly broad. For example excluding all files in the `/images/` folder would be counterproductive.

Exclusions

There may be certain images that you want to always load, regardless of where those images display on the page. Use case-sensitive, regular expressions to ensure that all images in a specific directory or of a particular type will always load. FEO will match exclusion patterns against SRC in attributes in the HTML page. Consider the examples shown below.

Examples of Excluding Images from On-Demand Image Loading

Scenario	Exclusion Pattern	Result
You need to always load the image bottomBanner.png. The image's URL is http://example.com/ads/bottomBanner.png	`/ads/`	Will always load *all* images contained in the ads directory.
	`bottomBanner\.png`	Will always load any image with a filename of bottomBanner.png, regardless of its path.
You want all .jpg images to load all the time.	`\.jpg$`	Will always load all images with an HTML IMG SRC attribute value that ends in .jpg.

Optimize CSS

FEO includes a set of techniques for optimizing cascading style sheets (CSS), which are bundled into the single Optimize CSS method. These techniques include Cookieless Static Assets, File Versioning, and DomainSharding. Gains resulting from optimizing style sheets will depend on which optional settings you enable. Review the table below to see how these benefits break out across optional settings.

Optional Settings

Enable zero or more of the optional settings defined below.

ⓘ **Note:** The HTML5 Advanced Cache setting in Optimize CSS is incompatible with the Critical CSS for Mobile optimization and the Edge CSS optional setting in EdgeStart. A policy may only include one of these three.

Optional Settings for CSS Optimizations

Setting	Description	Benefit
Convert Imports to Links	• Extracts imported CSS files and replaces them with CSS links or HTML5 Advanced Cache items.	Streamlines load order.

Setting	Description	Benefit
HTML5 Advanced Cache	• Uses HTML5's local storage to extend the CSS files' caching duration. • Retrieves CSS files from the server only when they are missing from the user's local cache.	Reduces HTTP requests.
Adaptive Consolidation	• Is only available when HTML5 Advanced Cache is enabled. • When none of the needed CSS files are available locally, FEO will make one request to download all the resources as a consolidated archive.	Reduces HTTP requests.
Asynchronous CSS Loading	• Processes style sheets without delaying other page resources from downloading.	Streamlines load order.
Image Compression	• Removes unnecessary metadata from images. • Uses techniques specific to each image's format to ensure lossless compression. • Reduces download size without changing the images' appearance.	Pares down page size.

Setting	Description	Benefit
Use Web Resolution	• By encoding images as text, eliminates the browser's need to request the image separately from the CSS file. • Embeds small images as data URIs into CSS.	Pares down page size.
Minification	• Removes unnecessary characters such as line breaks,white space, and comments from .css files. • Rewrites code using short-form syntax wherever possible. For example, FEO will abbreviate a color value of #FFFFFF to #FFF.	Pares down page size.
Small Image Embedding	• By encoding images as text, eliminates the browser's need to request the image separately from the CSS file. • Embeds small images as data URIs into CSS.	Reduces HTTP requests.

Exclusions

For any style sheet you do not wish to optimize, enter a text-based pattern to match against the style sheet's URL. Or enter a pattern to exclude an entire directory of style sheets. Use regular expressions for exclusion patterns, which can match on any part of a URL for that URL to be excluded.

Examples of Excluding Styles from CSS Optimization

Scenario	Exclusion Pattern
You do not want to optimize a specific style sheet, for example **IE6OneOffs.css**	`IE6OneOffs\.css`

Scenario	Exclusion Pattern
You want to leave alone all of the style sheets contained in one directory, for example / **IEstyles**	`/IEstyles/`

Optimize Images

FEO includes a set of techniques for optimizing images, which are bundled into the single Optimize Images method. These techniques include Cookieless Static Assets, File Versioning, and Domain Sharding. Additional gains resulting from optimizing images will depend on which optional settings you enable.

Optimizing images pares down page size and can reduce render times significantly.

ⓘ **Note:** This optimization can be used to reduce image resolution.

Optional Settings

Enable zero or more of the optional settings defined below.

Setting	Description	Benefit
Image Compression	• Removes unnecessary metadata from images • Uses techniques specific to each image's format to ensure lossless compression. • Required to enable browser-specific optional settings.	
Use the JPEG 2000 Image Format for Safari	• Is only available when Image Compression is enabled. • JPEG 2000 is a wavelet-based image format that provides both lossy and lossless compression. • Multiple sizes/ resolutions can be	Pares down page size.

Setting	Description	Benefit
	served from a single JPEG 2000 file. • Supports Region Of Interest (ROI) encoding, which allows higher resolution in a defined portion of the image. • FEO will only use JPEG 2000 formatting when optimizing for the Safari browser.	
Use the JPEGXR Image Format for IE 10+	• Is only available when Image Compression is enabled. • JPEGXR is a Microsoft-backed image format comparable to WebP in file size. • FEO will only use JPEGXR formatting for supported browsers,which are limited to Internet Explorer versions 10.0 and higher.	Pares down page size.
Use the WebP Image Format for Chrome	• Is only available when Image Compression is enabled. • WebP is a Google-created image format that provides both lossless and lossy compression for images on the web. • WebP lossless images are 26% smaller in size compared to PNG files.	Pares down page size.

Setting	Description	Benefit
	• FEO will only use WebP formatting when optimizing for desktop and Android versions of the Chrome browser.	
Use Web Resolution	• Is only available when Image Compression is enabled. • Performs lossy compression on images by reducing the number of colors, resolution, and accuracy to values suitable for digital displays.	Pares down page size.
Resize to HTML Dimensions	• Scales down each image to the dimensions specified in the image tag when the tag's width and height values are smaller than the image's actual dimensions. • See *Cautions*, for information about when *not*to use this setting.	Pares down page size.
Responsive Images	• Creates multiple scaled-down versions of images. • Identifies device's screen width and height. • Serves the browser images sized for the requesting devices' screen size.	Pares down page size.

Setting	Description	Benefit
	• Reduces load time when serving images to smaller screens without the need to maintain multiple image sizes on your origin. • See *Considerations for Responsive Images*, for more information.	
Use Lower Resolution on 2G/3G	• Is only available when Responsive Images is enabled. • Displays lower-quality images when FEO detects that the requesting browser is using a 2G or 3G cellular connection.	Pares down page size.
Specify Image Dimensions	• Adds image width and height values to image tags when both attributes are absent from the original HTML file. • See *Cautions*, for information about when *not*to use this setting.	Pares down page size.

Exclusions

To exclude an image, first select from the drop down menu to exclude images from JPEGXR formatting, WebP formatting, JPEG 2000 formatting, or all image optimizations. Then click the **Add Exclusion** button and type a regular expression consisting of all or part of the image filename. Or enter a directory name to exclude its contents. Exclusion patterns can match on any part of the URL to be excluded.

Examples of Excluding Images from Optimizations

Scenario	Exclusion Pattern
You do not want to optimize any .jpg or .jpeg files.	`\.jpe?g$`

Scenario	Exclusion Pattern
You do not want to optimize any images that are part of the catalog.	`/catalog/`
You do not want to optimize any images matching a specific query string on the multiimage.php script.	`multiimage\.php\?image=1`

Cautions

Using optimization settings that alter dimensions is most effective when the page's optimized images are used consistently in the original HTML. There are two cases when these settings are not recommended:

- If the same origin image is used in several places but with different dimensions hard-coded into the HTML, the images may appear stretched in the rendered page. If this is the case, do not select the **Resize to HTML** setting when configuring your image optimizations.
- If images change dynamically in unintended ways, do not select the **Specify Image Dimensions** setting.

Considerations for Responsive Images

The following points apply specifically to the Responsive Images optional setting:

- A client-side script determines the screen size, not Device Characterization.
- Be aware that Retina devices have the same screen width as non-Retina devices, so FEO serves the same scaled-down image size to both types of screens.
- You might want to use Adaptive Image Compression (AIC) instead of or in addition to Responsive Images in some circumstances:
  - Only images that appear on the URL constant page will be optimized. Responsive Images will not apply to images that change dynamically (for example, product images).
  - Responsive Images prioritizes image quality over load time. If network conditions are a more important concern than image quality, AIC might be a better choice.

Optimize JavaScript

In addition to narrowly-targeted optimization methods such as Asynchronous JavaScript and JavaScript Pre-Execution, FEO bundles a set of optimizations for JavaScript into the Optimize JavaScript method. These techniques include Cookieless Static Assets, File Versioning, and Domain Sharding. Gains resulting from optimizing JavaScript will depend on which optional settings you enable. Review the table below to see how these benefits break out across optional settings.

Optional Settings for JavaScript Optimizations

Setting	Description	Benefit
HTML5 Advanced Cache	• Uses HTML5's local storage to extend the JavaScript files' caching duration. • Retrieves JavaScript files from the server only when they are missing from the user's local cache.	Reduces HTTP requests.
Adaptive Consolidation	• Is only available when HTML5 Advanced Cache is enabled. • When none of the needed JavaScript files are available locally, FEO will make one request to download all the resources as a consolidated archive	Reduces HTTP requests.
Streaming Consolidation	• Stores and evaluates the individual resources as they stream in instead of waiting until the entire archive finishes downloading.	Streamlines load order.
Inline JavaScript	• Embeds code from external JavaScript files into HTML files that have a maximum age (TTL) of two hours or greater. • Is best used for small scripts; use the **Do Not Inline Scripts Bigger Than** field to set a threshold for maximum size of files	Reduces HTTP requests.

Setting	Description	Benefit
	to be inlined (default: 4096 Bytes). • Eliminates the need to make HTTP requests for the external file. • Stores inlined JavaScript in cache. – Is most effective when the HTML5 Advanced Cache setting is also enabled. When this is the case, FEO prevents inlining of repeated page views by setting a cookie in the domain's cookie space.	
Minify External JavaScriptandMinify Internal JavaScript	• Removes from JavaScripts any unnecessary characters such as line breaks, white space, and comments. • Can be applied to internal JavaScripts, external JS files, or both.	Pares down page size.

Exclusions

Use the Optimize JavaScript exclusion to prevent FEO from applying any optimizations to the file(s) you specify. To exclude a JavaScript file, enter an exclusion pattern consisting of all or part of the JavaScript filename. For a URL to be excluded, the exclusion pattern can match on any part of that URL. Consider the example below.

Example of a JavaScript Optimization Exclusion

Scenario	Exclusion Pattern
You want to shield all files in the catalog directory from all JavaScript optimizations.	`/catalog/`

Page Prefetching

⚠ **Important:** This FEO feature will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations.

Page Prefetching speeds up the load of the next page a user is likely to visit. You apply your knowledge of usage patterns on your sites to indicate which page a user is most likely to visit from a given starting page. Page Prefetching starts loading the resources for the second page while the user is interacting with the first page. If the user then navigates to the second page, it will load faster.

You define the page to prefetch by configuring the *Prefetch Link*, a regular expression to map against links contained in the requested page's HTML. This regular expression matches on all or part of the URL of the page to be prefetched.

How It Works

If FEO receives a request that matches the URL match criteria of a policy that contains the Page Prefetching optimization, it checks the prefetch link.

FEO then searches the page's content for a link matching the prefetch link regular expression. Once the requested page finishes loading, it prefetches the first matching URL found and begins loading any resources in the prefetched page.

ⓘ **Note:** Chrome browsers behave differently. Instead of prefetching the linked page, Chrome prerenders it in a hidden tab.

Optimize Target Linked Pages

With one exception, FEO will preload optimized resources (images, scripts, style sheets, etc.) only if the page to prefetch has already been transformed. Therefore, prefetched pages should themselves be optimized.

Check that all pages that match the prefetch link pattern are captured in at least one FEO policy's URL matches. It is only necessary that those pages be optimized; you do not need to apply the Page Prefetching optimization to the linked pages.

Exception: Because Chrome browsers use prerendering instead, it is *not* necessary to optimize the linked page for requests from Chrome browsers. Even if the target page is not optimized, Chrome browsers will prerender it.

Examples

For example, to prefetch the first link on News.com's home page that is page in its Technology directory, you would do the following:

How to

1. Define a policy with a URL match such as `http://www.news.com/*` to match on www.news.com.
2. Define a prefetch-link pattern representing the complete URL. If the links to technology pages look like this `<a href="http://www.news.com/tech/index.html">` , then using a regular expression such as `href="(/tech/)"` will result in prefetching a page in the tech directory. Enter this link pattern in the **Prefetch Link** text field.

Cautions

Be careful about the syntax for prefetch patterns. Follow these rules:

- Matches are case-insensitive regular expressions that must match the source HTML.
- Use parentheses to indicate the link itself.
- Below are some examples of source code links and corresponding prefetch patterns:

If source code's link looks like this...	...then use this prefetch pattern:

`<a href="LINKNAME"> <iframe src="LINKNAME">`	`href="(PREFETCH-PATTERN)"`
	`src="(PREFETCH-PATTERN)"`

- Use a pipe (|) to separate patterns for multiple URLs.
- To match against the link's HTML tags, put additional criteria outside of the parentheses.

Also remember the following:

- For any given request, FEO prefetches only one page.
- FEO prefetches only the *first* matching URL that appears in the requested page's HTML. This is true even if the first URL is in a line that is commented out and therefore doesn't appear on the start page.
  - Be sure to account for URLs in comments when determining what pattern to use.
  - You can deliberately plant a URL in a comment near the top of a page to force FEO to prefetch a URL that otherwise would not be the first matching URL. Be sure the comment is not inside any JavaScript or CSS elements (the `<script>` and <style> tags).

Best Practices

To maximize the effectiveness of this optimization method, you need to evaluate the activity on your site. Use your site's analytics to determine visitor patterns, so that you only prefetch frequently visited pages.

Resource Prefetching

Important: This FEO feature will reach end of life in March 2021. You should disable this option before support ends if you want to test your revised policy. Features that reach end of life before they are disabled will stop performing optimizations.

With Resource Prefetching, JavaScript and CSS files are cached on the user's browser before they are requested.

Resource Prefetching is especially useful when a configuration has multiple policies, with each policy representing a group of similar pages. Often, visitors to a web site to access a particular sequence of web pages. When a visitor first accesses one page, Resource Prefetching loads CSS and JavaScript resources used by other pages belonging to the other policies in the same FEO configuration. Resources belonging to other pages do not begin loading until after the current page is loaded, so it does not impact the performance of the current page. These resources are stored in the browser's HTML5 cache.

Consider a configuration with three policies: one for the home page, one for category pages, and one for product pages. Suppose the pages associated with these policies have the following CSS and JavaScript resources:

Policy	CSS Resources	JavaScript Resources
Home Page	**main.css** **home.css**	**login.js** **salecountdown.js**
Category Page	**main.css** category.css	**login.js** saleslides.js
Product Page	**main.css** **home.css** product.css sale.css	**login.js** saleslides.js cart.js productnavigation.js search.js

With Resource Prefetching enabled on the above policies, a first-view visit to the Home Page prompts the Home Page resources (marked in boldface) to begin loading first. Once the page load is complete, the remaining resources associated with Category and Product Pages will begin to load.

When the visitor navigates to a Category Page and a Product Page, the CSS and JavaScript resources will already be in the browser's HTML5 cache. This reduces asset download and improves the performance of the first-view visits to these pages.

Cautions

If there are few unique CSS and JavaScript resources across pages belonging to different policies (i.e., CSS and JS is almost all common), then the effectiveness of this optimization is reduced.

Best Practices

To ensure greater consistency and reliability, use a URL constant to seed the HTML5 cache with the same resources every time, regardless of which of the policy's pages is the first to request content. Do note that using a URL constant overrides the first-request behavior for all selected optimization in the policy, not just Resource Prefetching.

For more information, review the *URL Constant* description. For usage directions review *The Edit Optimization Policies Page* and the *Use URL Constant As Policy Benchmark* description.

Optional Settings

There are two optional settings:

- Select the **Prefetch CSS** checkbox to store style sheets in the browser's HTML5 cache.
- Select the **Prefetch JavaScript** checkbox to store JavaScript files in the browser's HTML5 cache.

If neither checkbox is selected, this effectively disables Resource Prefetching, even if this optimization is in the selected methods column for the policy.

Exclusions

You may use any combination of three types of exclusions for the Resource Prefetching optimization method. These exclusion options provide a more targeted way to prevent caching rogue files than using the URL constant.

Exclude Specific CSS Files from Prefetch and Exclude Specific JavaScript Files from Prefetch

Use these exclusions to prevent specific style sheets and JavaScript files from caching in the browser's local storage.

How to Exclude CSS and JavaScript Files from Prefetch

How to

1. Go to the Resource Prefetching **Exclusions** tab, select the type of exclusion from the drop-down menu. and select the **Add Exclusions** button.
2. In the **Click to edit text** field, use a regular expression to indicate the file(s) to exclude.
3. Type **Enter** on your keyboard or select **Add Exclusions** again.
4. Continue to enter exclusions as desired.

Exclude Policy Resources from Prefetch

Use the existing match criteria of other policies in your FEO configuration to exclude file caching in batch. For example, say you have a policy, Expiring Deals, designed to optimize product pages that have very short TTLs. The policy's match criteria is `http://www.example.com/products/closeouts/*`. Excluding the Expiring Deals policy from Resource Prefetching will keep all CSS and JavaScript files out of the browser's cache.

How to Exclude All of Another Policy's Resources from Prefetch

How to

1. Go to the Resource Prefetching Exclusions tab, select Exclude Policy Resources from Prefetch from the drop-down menu, and select the Add Exclusions button.
2. Select the Click to edit text field.
3. This displays a drop-down menu auto populated with the names of the other policies in your FEO configuration.
4. Select the desired policy of exclusion.
5. Type Enter on your keyboard or select Add Exclusions again.

What you should see

ⓘ **Note:** Though you can go back and forth among different exclusion types and hop from working with one optimization method to the other, be certain to select the **Save** button at the bottom of the page in order to preserve your policy changes, and then select **Save Configuration**.

The Front-End Optimization API v1

The Front-End Optimization (FEO) API provides a way to use FEO functionality programmatically rather than through FEO's Configuration Manager.

FEO accelerates web pages by creating modified versions of the HTML and supporting resources that can render more quickly. The techniques FEO uses to accelerate pages are called *optimization methods* (or just optimizations). Some optimization methods are specific to resource files such as images, JavaScript, and style sheets. Other optimizations adjust the prioritization and queuing of rendering activities.

Who should use this API

This API is intended for developers who want to automate FEO management. To use this API, you should be familiar with your existing FEO configuration and the content it's optimizing, as well as the parent product configuration. You should know how to manage FEO configurations in FEO's Configuration Manager (accessible from the **Configure** menu from a Property Manager page specific to a property that has FEO enabled in its configuration), including activation and purging. It is important to have a general understanding of how FEO works, including page analysis, creating transformation bundles, optimizing pages and resources, and delivering origin vs. optimized content.

If you're not familiar with those concepts, have a look at the *Front-End Optimization User Guide* before you continue.

Purge a configuration

Use this API to purge an FEO transformation bundle.

The FEO API deletes all transformation bundles for an `assetName`. The network checks the status of transformation bundles about every five minutes. If it finds that a transformation bundle has been deleted, the FEO analyzer performs a new analysis and generates a new configuration bundle. If resources have been added or modified, the FEO transformer re-optimizes them.

Request

POST `/feo/v1/purge`

Object type: *Purge*

Content-Type: `application/json`

Request body:

```
{
    "assetName": "My Main Site",
    "network": "S",
    "comments": "Spring catalog updates on main site"
}
```

Usage example

Assume that the product catalog on your website updates weekly. You use the *Fast Purge API* to purge the parent product's configuration, to remove any stale origin content. When this purge finishes, you can then run an FEO purge to remove the transformation bundles for each of your properties.

1. Use PAPI get a list of your properties for your applicable `contractId` and `groupId`.

2. Store the `propertyName` for each property you want to purge.

```
GET /papi/v0/properties?groupId=grp_51529&contractId=ctr_3-LPONXD

{
  "properties": {
    "items": [
      {
        "accountId": "act_Z-8-SA8DFS",
        "contractId": "ctr_3-LPONXD",
        "groupId": "grp_51529",
        "latestVersion": 3,
        "productionVersion": 3,
        "propertyId": "prp_256737",
        "propertyName": "www.site.example.com", <= Store this for FEO
purge
        "stagingVersion": null
      },
      {
        "accountId": "act_Z-8-SA8DFS",
        "contractId": "ctr_3-LPONXD",
        "groupId": "grp_51529",
        "latestVersion": 2,
        "productionVersion": null,
        "propertyId": "prp_581213",
        "propertyName": "jsmith_testing",
        "stagingVersion": null
      },
      {
        "accountId": "act_Z-8-SA8DFS",
        "contractId": "ctr_3-LPONXD",
        "groupId": "grp_51529",
        "latestVersion": 4,
        "productionVersion": 4,
        "propertyId": "prp_52887",
        "propertyName": "www.random.example.com", <= Store this for FEO
purge
        "stagingVersion": null
      }
    ]
  }
}
```

3. Use FEO purge for the first stored `propertyName`, using it as the `assetName` in the request body. Set the Akamai `network` where the purge should take place `S` for staging or `P` for production, and add `comments` to explian why you performed the purge.

```
POST /feo/v1/purge/
Content-Type: application/json
```

```
{
  "assetName": "www.site.example.com",
  "network": "S",
  "comments": "Spring catalog updates on main site"
}
```

4. Typically, the initial response will state, `"errorMessage": "Change pending"`. The FEO API doesn't provide an indication of when the request has finished. Make an additional request using the same `assetName`. If you see `"errorCode": "401"` in the response, wait a little while and try the operation again. Once you see `"errorCode": "400"`, the purge has completed successfully.

```
{
  "assetId": "12345670",
  "errorCode": "400",
  "errorMessage": "Successful operation"
}
```

Note: Both of these points apply at this phase of the purge:

- You can only have one FEO purge request active on a property at a time.

- The `errorCode` values returned in a response are not the same as standard HTTP status codes. They're unique to this API and reflect the success or error of an FEO purge request. See *Errors* for more information.

5. Repeat steps 3-4 for each additional stored `propertyName` values that you want to FEO purge.

```
POST /feo/v1/purge/
Content-Type: application/json

{
  "assetName": "www.random.example.com",
  "network": "S",
  "comments": "Removals from the spring catalog on main site"
}
```

The Purge data objects

This section provides details about the purge data objects used in requests and responses with this API.

The request object

When making a purge request, you specify the property and the network, (staging or production). By default, this purges the transformation bundles associated with whichever FEO configuration version is currently active in thatnetwork. Optionally, you can specify a different FEO configuration version. A comment is also required so you can recognize the purpose of a given purge action.

Below is an example of the JSON and a table listing the data members for the *JSON request*.

```
{
  "assetName": "www.feo-test-shverma.com",
  "network": "P",
  "comments": "my comments"
}
```

Member	Type	Required	Description
`assetName`	String	✓	The Property Name (as configured in Property Manager) for which you wish to issue the FEO Purge request
`network`	Enumeration	✓	The environment to which you wish to issue the FEO Purge request, either `P` for production (default) or `S` for staging.
`comments`	String	✓	A description of the reason for the purge, for tracking purposes

The response object

After you initiate a purge, you'll receive a JSON response object. Below is an example and a table describing its members.

```
{
  "assetId": 93912312,
  "errorCode": 407,
  "errorMessage": "No active version"
}
```

Member	Type	Required	Description
`assetId`	Number	✓	A unique integer identifier that is functionally equivalent to the `assetName` specified in the request
`errorCode`	Number	✓	This is a unique numeric code that represents the status of the purge request. See *errorCode* values for more information.
`errorMessage`	String	✓	A text description of the error code; returns `null` on success when `errorCode` is `400`

errorCode values

This table describes the range of possible numeric status codes for the `errorCode` member returned in the response object.

Code	Message
400	Successful operation

Code	Message
401	Change pending
402	Database error
404	Incorrect property name
407	No active version
408	No ARL for asset
409	No beta permission
410	No comment provided
411	No configuration for asset
413	No property access
415	No versions
417	Submit failed
418	Unexpected exception
419	Version cannot be purged

See *Errors* on page 81 for more details

Errors

This section provides details for the API's set of possible error scenarios, and strategies to address them.

401: change pending

This error code indicates that the version you are attempting to purge already has a pending Activation, Deactivation or Purge Transformations request in progress.

Request

```
POST /feo/v1/purge/
Content-Type: application/json

{
  "assetName": "www.mysite.com_pm",
  "network": "S",
  "comments": "Purging a property with a purge already in progress"
}
```

Response

```
{
  "assetId": "1760254",
  "errorCode": "401",
  "errorMessage": "Change pending"
}
```

Resolution

Wait for a little while for the existing request to complete, before issuing another purge request.

404: incorrect property name

The property specified in `assetName` was not found.

Request

```
POST /feo/v1/purge/
Content-Type: application/json

{
  "assetName": "www.myiste.com_pm",
  "network": "P",
  "comments": "Purging a property with a typo in the assetName"
}
```

Response

```
{
  "assetId": "1600833",
  "errorCode": "404",
  "errorMessage": "Did not find asset www.myiste.com_pm"
}
```

Resolution

Ensure that the value of `assetName` corresponds to a valid property.

407: no active version

This error code indicates that no active FEO configuration version exists for the network you are attempting to purge.

Request

```
POST /feo/v1/purge/
Content-Type: application/json

{
  "assetName": "www.mysite.com_pm",
  "network": "P",
  "comments": "Purging a property with no active Production version."
}
```

Response

```
{
  "assetId": "1032590",
  "errorCode": "407",
  "errorMessage": "No active configuration"
}
```

Resolution

Activate an FEO version within the specified network (Staging or Production) and/or ensure that the `network` parameter is correct.

410: no comment provided

This error code indicates that the `comments` field was missing from the request.

Request

```
POST /feo/v1/purge/
Content-Type: application/json

{
  "assetName": "www.mysite.com_pm",
  "network": "P"
}
```

Response

```
{
  "assetId": "1634446",
  "errorCode": "410",
  "errorMessage": "No comment provided"
}
```

Resolution

Make sure to include the `comments` parameter in the request.

411: no configuration for asset

This error code indicates that no FEO configuration versions exists for the specified property.

Request

```
POST /feo/v1/purge/
Content-Type: application/json

{
  "assetName": "www.mysite.com_pm",
  "network": "P",
  "comments": "Purging a property with FEO versions defined."
}
```

Response

```
{
  "assetId": "1910270",
  "errorCode": "411",
  "errorMessage": "Did not find Config for asset 1910270"
}
```

Resolution

Create and activate an FEO configuration version for the specified property.

Notice

Akamai secures and delivers digital experiences for the world's largest companies. Akamai's Intelligent Edge Platform surrounds everything, from the enterprise to the cloud, so customers and their businesses can be fast, smart, and secure. Top brands globally rely on Akamai to help them realize competitive advantage through agile solutions that extend the power of their multi-cloud architectures. Akamai keeps decisions, apps, and experiences closer to users than anyone — and attacks and threats far away. Akamai's portfolio of edge security, web and mobile performance, enterprise access, and video delivery solutions is supported by unmatched customer service, analytics, and 24/7/365 monitoring. To learn why the world's top brands trust Akamai, visit *www.akamai.com*, *blogs.akamai.com*, or *@Akamai* on Twitter. You can find our global contact information at *www.akamai.com/locations*.

Akamai is headquartered in Cambridge, Massachusetts in the United States with operations in more than 57 offices around the world. Our services and renowned customer care are designed to enable businesses to provide an unparalleled Internet experience for their customers worldwide. Addresses, phone numbers, and contact information for all locations are listed on *www.akamai.com/locations*.

Published 9/2020