MEDIA FILE FORMAT AND MARKETING AGREEMENT

This Media File Format and Marketing Agreement (the "Agreement") is entered into as of this 18th day of July, 1997 (the "Effective Date") by and between MICROSOFT CORPORATION, a Washington corporation located at One Microsoft Way, Redmond, WA 98052 ("Microsoft") and PROGRESSIVE NETWORKS, INC., a Washington corporation located at 1111 Third Avenue, Suite 2900, Seattle, WA 98101 ("PN").

RECITALS

A. The parties entered into an "Agreement between Microsoft and Progressive Networks on Media Streaming Technology" on June 17, 1997 (the "License and Investment Agreement");

B. In Section 5 of the License and Investment Agreement, the parties agreed to enter into this separate Agreement governing, among other things, their joint development and support of a media file format to be included in their respective products;

C. The parties wish to enter into this Agreement, subject to all its terms and conditions.

NOW, THEREFORE, in consideration of the mutual covenants contained herein the parties agree as follows:

AGREEMENT

1. DEFINITIONS

In addition to the terms defined elsewhere in this Agreement, the following terms, when used herein, shall have the following meanings:

1.1 **"ASF"** means the final specification for a multimedia streaming file format to be jointly developed by Microsoft, PN and third parties from the Current ASF Draft and which is publicly released to third parties.

1.2 **"Confidential Information"** means: (i) any trade secrets relating to either party's product or service plans, designs, costs, prices, customer names, finances, marketing plans, business opportunities, personnel, inventions, software programs, proprietary information, research, development or know-how; and (ii) the specific terms and conditions of this Agreement. "Confidential Information" shall not include information that: (i) is or becomes generally known or available, whether by publication, commercial use or otherwise, without restriction on disclosure and through no fault of the receiving party; (ii) is known and has been reduced to tangible form by the receiving party at the time of disclosure and is not subject to restriction; (iii) is independently developed or learned by the receiving party without reference to any Confidential Information of the disclosing party; (iv) is lawfully obtained from a third party that has the right to make such disclosure; and (v) must be disclosed in response to a valid order by a court or other governmental body, is otherwise required by law, or is necessary to establish the rights of either party under this Agreement. The exceptions described herein apply only to the parties' obligations under Section 7 of this Agreement and do not in any way alter any other obligations or restrictions, such as those arising from copyright, patent or trade secret law.

1.3 **"Current ASF Draft"** means the preliminary specification for a multimedia streaming file format that is Microsoft Confidential Information and is attached hereto as Exhibit A.

1.4 **"DirectDraw"** means Microsoft's multi-media software technology for 2D graphics operations, which are exposed by the 2D application programming interfaces in the collection of multi-media services known as the Microsoft DirectX technology.

1.5 "DirectShow" means Microsoft's DirectX client streaming technology for audio, video and other data formats that are exposed by a set of application programming interfaces and filter technologies in the DirectX technology.

1.6 "DirectX" means the Microsoft's multi-media services and related application programming interfaces that are included in the Win32 Systems.

1.7 "Internet Explorer" means the standard version of Microsoft Internet Explorer 4.0 for Microsoft Windows 95 and Windows NT, including all upgrades, versions, and successors thereto commercially released by Microsoft during the term of this Agreement.

1.8 "PN Clients" means all of PN's RealAudio and RealVideo clients or "players" for Win32 Systems, including but not limited to the PN Player.

1.9 "PN Player" means the version of PN's RealAudio and RealVideo client or "player" software, version 4.0, including Upgrades thereto and including all non-English versions thereof for languages supported by PN, that Microsoft distributes with Internet Explorer.

1.10 "RMFF" means the current native file format included in PN's RealAudio and RealVideo 4.0 products.

1.11 "Upgrades" means all upgrades, bug fixes, version and successors commercially released during the term of this Agreement.

1.12 "Win32 Systems" means Microsoft Windows 95 and Microsoft Windows NT and all upgrades, versions and successors thereto.

2. COOPERATION ON FORMATS AND PROTOCOLS

The parties agree to work together in good faith throughout the term of this Agreement on the definition of formats and protocols for streaming media. As part of this cooperation, the parties shall use reasonable efforts to make their respective products that include streaming media capabilities compatible and interoperable with each other. This commitment, however, shall not limit either party's rights to innovate and introduce value-added products and features not supported by the other parties' products.

3. OWNERSHIP AND SUPPORT OF ASF

3.1 Development and Ownership of ASF.

(a) The parties agree to work together and with third parties in good faith to jointly develop ASF from the Current ASF Draft by no later than September 30, 1997. The parties must jointly agree on the contents of ASF before it is publicly released. In the event the parties so agree, the parties shall be publicly listed as co-authors of ASF and shall continue working in good faith thereafter on new updates and versions of the specification during the term of this Agreement. In the event the parties do not agree on ASF by September 1, 1997, then the parties shall engage in appropriate executive level discussions to determine whether they can reach a compromise solution on the definition on ASF. If the parties are unable to reach a compromise solution by September 30, 1997, PN shall not be deemed a co-author of such specification and Microsoft shall own all right, title and interest thereto, and this Agreement shall immediately terminate as provided in Section 9.

(b) The parties agree that Microsoft shall own and, except as described below, control modification of the Current ASF Draft, ASF and new updates and versions thereof, but shall involve PN in any process that Microsoft defines in consultation with PN to

obtain input on the specification from interested parties. Accordingly, PN hereby irrevocably conveys and assigns to Microsoft, its successors and assigns, all right, title and interest in and to any intellectual and proprietary rights in PN's contributions to ASF and new updates and versions, except for those modifications as described in sub-section (c) below that are suggested by PN in good faith and rejected by Microsoft, which shall be owned by PN. Microsoft hereby grants PN a non-exclusive, perpetual, irrevocable, worldwide, royalty-free right and license, under Microsoft's intellectual property rights, to use, copy, modify, create derivative works from, license and distribute the ASF specification and new updates and versions thereof and modified versions and derivative works thereof in the PN Player and Upgrades thereto.

(c) In the event that PN wishes to modify or create derivative works of ASF, then PN shall first propose in good faith to Microsoft that such modifications be included in ASF and/or a new update or version thereof as a PN contribution to the specification to be owned by Microsoft and licensed to PN as described in this Section." If Microsoft rejects such proposed modifications, then the parties shall engage in appropriate executive level discussions to determine whether they can reach a compromise solution to include the proposed modification in ASF. If the parties are unable to reach a compromise solution within a reasonable period of time, PN may make the modification itself to the ASF specification provided that PN does not refer to the modified specification as "ASF" or some other name that is confusingly similar to ASF and PN includes a prominent legend on the front of every copy of the modified specification indicating that it has been modified from the original, copyrighted version available from Microsoft. During the negotiations and thereafter, if Microsoft elects to not incorporate PN's extensions to ASF, Microsoft will continue to distribute the PN Player as provided in this Agreement.

3.2 Support for ASF. PN agrees that it will commercially release an Upgrade to the PN Player that includes a full implementation of ASF within 180 days after the release of the ASF specification by Microsoft (the "Release Date"). PN shall treat ASF as the default and preferred file format for its products after the Release Date and shall not release further updates to RMFF after the date that it commercially releases Upgrades to the PN Player with support for ASF.

4. SUPPORT OF DIRECTSHOW AND DIRECTDRAW; LICENSE TO CONFIDENTIAL INFORMATION REGARDING MICROSOFT CODEC AUTHENTICATION TECHNOLOGY

4.1 PN agrees to integrate support for DirectShow into Upgrades to the PN Clients as soon as practicable and no later than March 31, 1998 and to continue such support throughout the term of this Agreement. PN further agrees to integrate support for DirectDraw into Upgrades to the PN Clients as soon as practicable and no later than September 30, 1997 and to continue such support throughout the term of this Agreement. As used in this Section, the term "support" means that PN shall implement such changes in the software and documentation portions of the PN Clients so that the PN Clients, including Upgrades thereto, primarily use DirectShow and DirectDraw for displaying media content including support for: (a) a DirectShow source filter that exposes a PCM pin for decoded audio and an RGB pin for decoded video, (b) the publishing of all video and audio PN codecs as DirectShow codecs, and (c) utilization of the DirectShow IMultiMediaStream interface for video and audio codecs. Notwithstanding the foregoing, if PN determines in good faith that there is a reason which prevents PN from primarily using DirectShow or DirectDraw in the manner described, then the parties shall engage in good faith executive level discussions to determine whether they can reach a compromise solution so that PN can continue to primarily use DirectDraw and DirectShow. PN further agrees that if for any reason they plan to support any programming interface published by Sun Microsystems or Netscape which performs substantially the same function as DirectDraw or DirectShow, they will promptly contact Microsoft to engage in good faith executive level discussions. If the parties are unable to reach a compromise solution within a reasonable period of time, PN may choose to use an alternative to DirectShow or DirectDraw in that particular

instance. In this event and given that executive level discussion in good faith has taken place, Microsoft agrees to continue to distribute the PN Player as provided in this Agreement. Microsoft agrees that it will continue to distribute the PN Player during the good faith executive level discussions.

4.2 Microsoft agrees to provide PN with prompt access after the Effective Date to certain Confidential Information regarding Microsoft's implementation of a codec authentication technology in Microsoft NetShow. Microsoft hereby grants PN a non-exclusive, royalty-free, worldwide right and license under Microsoft's intellectual property rights to such Confidential Information for the sole purpose of implementing the same or a similar authentication technology in the PN Player and Upgrades thereto.

5. ACTIVE DESKTOP CHANNEL

The parties understand that beginning with version 4.0 of Microsoft Internet Explorer, vendors will have the ability to "push" digital media from Web-based servers to end users of Internet Explorer through various "channels." Microsoft plans to provide a listing to end users of certain channels available from vendors with whom Microsoft has marketing relationships. These channels shall be organized into different tiers on the Internet Explorer desktop. Microsoft agrees that as soon as and for so long as it maintains this "push" functionality in Internet Explorer and the various "tiers" described in this Section during the term of this Agreement, it shall include a PN channel in the "gold" or second tier of channels that Microsoft lists in Internet Explorer 4.0 and future versions thereof. This Section shall not become effective unless and until PN signs Microsoft's standard agreement governing the further rights and obligations of the parties with respect to such push channel marketing and distribution.

6. RIGHTS AND OBLIGATIONS RELATED TO PN PLAYER

6.1 Delivery and Acceptance of PN Player and Upgrades. PN shall deliver ten (10) copies of the PN Player, including all non-English language versions, to Microsoft within five (5) days after the Effective Date. The copies of the PN Player shall be deemed accepted by Microsoft upon delivery unless Microsoft notifies PN in writing (the "Rejection Notice") within ten (10) days thereafter that it has determined in good faith that the PN Player contains software viruses or bugs that materially effect the performance of the PN Player. In such event, the parties shall discuss additional correction period in good faith, not to exceed thirty (30) days for PN to deliver conforming software. If the parties are unable to resolve the issue in good faith within such 30 day period, Microsoft shall have an option to allow PN an additional ten (10) day period within which to deliver ten (10) new copies of the PN Player that do not have such deficiencies or to return the software and terminate this Agreement in its entirety. PN shall deliver Upgrades to the PN Player, including all non-English language versions, to Microsoft no later than the date that it makes such Upgrades available to any third party. PN also shall include Microsoft in beta software programs that PN creates for customers and other users of the PN Player and Upgrades.

6.2 Player Requirements. All Upgrades to the PN Player shall include the following features: (a) they shall support PN RealAudio 1.0 and PN RealVideo 4.0 server software and Upgrades thereto when commercially available, (b) they shall be capable of streaming media from any Real Audio/Real Video compatible server, including those established and maintained by Microsoft, (c) they shall use DirectShow and DirectDraw when and as described in Section 4, and (d) they shall support ASF when and as described in Section 3. The parties understand and agree that the size of the PN Player and Upgrades is very material to Microsoft and that Microsoft wishes PN to make reasonable efforts throughout the term of the Agreement to minimize the amount of memory, disk storage, and download size required for the PN Player and Upgrades thereto. Accordingly, the parties agree that (a) the PN Player and Upgrades thereto shall not require more than 1.2 megabytes of download size, and (b) Microsoft may exclude or block certain features of the PN Player and Upgrades thereto to minimize the disk storage and download size.

6.3 Material Changes. PN shall obtain advance consent from Microsoft for any material changes to the PN Player and Upgrades thereto during the term of this Agreement. For purposes of this Section, the term "material change" shall include, but not be limited to, any change that substantively

effects the PN Player's ability to stream media, such as changes to or additions of any new or modified codecs, filters, streaming formats, datatypes, protocols, programming interfaces of any type, unicast and multicast features, and any other changes that substantively effect the core user experience of viewing streaming media. PN agrees that Microsoft shall have sole discretion whether to grant its consent to changes described in this Section 6.3. PN agrees to take reasonable steps to ensure that the DirectShow client software can steam any media that can be played by the PN Player throughout the term of this Agreement. The parties understand and agree that at the option of PN, if PN wishes to make material changes, such reasonable steps may include license and delivery of source code or binary components by PN to Microsoft for inclusion in DirectShow; the parties further agree that Microsoft may, at its discretion, grant its consent to such changes and enter into such license agreements and accept delivery of such code.

6.4 Non-Material Changes. PN shall obtain advance consent from Microsoft prior to making any (a) non-material changes to the user interface of the PN Player and Upgrades thereto, (b) changes to take advantage of Microsoft system enhancements, and (c) changes to utilize technologies that are not considered Material as defined in Section 6.3 that Microsoft is taking advantage of in its streaming media products. Microsoft shall not unreasonably withhold its consent to changes described in this Section 6.4 and shall be deemed to have given such consent if Microsoft does not notify PN of any objections within thirty (30) days from the date that PN gives Microsoft notice of such proposed changes.

6.5 License to PN Player and Upgrades. PN hereby grants to Microsoft, under the intellectual and proprietary rights of PN, a non-exclusive, irrevocable, worldwide, royalty-free right and license to (i) use, copy, license, broadcast, publicly display, transmit or otherwise distribute in any medium now known or hereafter devised (collectively, "Distribute") and have Distributed to and by third parties, binary versions of the PN Player and Upgrades thereto during the term of this Agreement and for so long as Microsoft continues to distribute the versions of Internet Explorer that have been released in beta or final form as of the date this Agreement expires or its terminated; and (ii) grant any and all of the rights set forth in this Section 6.5 to third parties, including the right to license such rights to further third parties.

The foregoing license grants include a license under any current and future patents owned or licensable by PN to the extent necessary to exercise any of the rights granted herein and to combine the PN and Upgrades thereto with any hardware and software.

6.6 Agreement to Distribute. Provided that PN has fully complied with its obligations under Sections 3, 4 and 6 of this Agreement, Microsoft shall (a) distribute one (1) copy of the PN Player with each copy of Internet Explorer that is distributed by Microsoft through its standard distribution channels, whether distributed on CD-ROM, through OEMs, or via on-line systems, and through any other channels and on any other media in and on which it distributes any other streaming media client, and (b) distribute one (1) copy of each new Upgrade to the PN Player in each new Upgrade of Internet Explorer, provided that such Upgrade to the PN Player has been delivered to Microsoft prior to the date that the Upgrade to Internet Explorer has been released in beta form. Notwithstanding the foregoing, PN understands and agrees that (a) Microsoft may exclude the PN Player and Upgrades from certain versions of Internet Explorer that are designed for minimal download time or are customized for end users of MSN, MSNBC, WebTV, CompuServe, AOL and other affiliates and commercial licensees of Microsoft, (b) Microsoft shall not be obligated to distribute the PN Player and Upgrades thereto in countries for which PN does not provide Microsoft a translated version, (c) Microsoft shall not be obligated to distribute the PN Player and Upgrades in the event a court of competent jurisdiction issues an injunction or similar order preventing further distribution of the PN Player and/or Upgrades thereto or determines that the PN Player and/or Upgrades thereto infringe upon the intellectual property rights of any third party, and that (d) certain original equipment manufacturers may elect not to include the PN Player and Upgrades in the software that they distribute with their computers and Microsoft shall be permitted to allow such original equipment manufacturers to exclude the PN Player and Upgrades, provided, that Microsoft shall offer the PN Player and Upgrades to such original equipment manufacturers as part of the basic or default operating system release.

MSS 000637
CONFIDENTIAL

6.7 Product Support. PN shall be solely responsible for providing end user support for the PN Player and Upgrades thereto and shall have discretion to create appropriate support policies for end user support.

6.8 Trademarks. PN hereby grants Microsoft a non-exclusive, royalty-free, fully paid up right and license to use the PN trademarks used in connection with the PN Player (the "Trademarks") to market the PN technology included in Internet Explorer. Microsoft agrees not to remove any Trademarks from the PN Player and Upgrades thereto that are reasonably inserted by PN and to use reasonable efforts to ensure that its distributors do not remove such Trademarks unless a court or administrative body of competent jurisdiction rules that such Trademarks or constituent elements thereof are infringing. Microsoft agrees to adhere to the PN Trademark Guidelines (as defined in that certain between the parties dated as of June 17, 1997) with respect to all other uses of the Trademarks; provided, that Microsoft's adherence to such guidelines shall be to the extent that such guidelines are consistent with Microsoft's general trademark practices as to third party trademarks, as modified from time to time. In addition, nothing herein shall circumscribe Microsoft's right to make fair, referential, comparative, descriptive or other references as permitted by applicable law.

6.9 No Further Obligation. Except as provided in Section 6.6, Microsoft shall have no obligation to market, promote or otherwise distribute the PN Player and Upgrades, either alone or as part of any Microsoft product or service. Nothing in this Agreement shall be construed as restricting Microsoft's ability to acquire, license, develop, manufacture or distribute for itself, or have others acquire, license, develop, manufacture or distribute for Microsoft, similar technology performing the same or similar functions as the PN Player and Upgrades thereto or to market or distribute such similar technology in addition to, or in lieu of, the PN Player and Upgrades thereto, except as provided in Section 6.6.

6.10 Effect of this Section. The parties agree that this Section 6 supersedes and replaces Section 2.5 of that certain License and Distribution Agreement between the parties dated as of August 8, 1996 (the "1996 Agreement"). The parties further agree that Section 3.2 of the 1996 Agreement is hereby deleted in its entirety.

7. CONFIDENTIALITY

Each party shall protect the other's Confidential Information from unauthorized dissemination and use with the same degree of care that such party uses to protect its own like information. Neither party will use the other's Confidential Information for purposes other than those necessary to directly further the purposes of this Agreement. Each party will use its best efforts not to disclose to third parties the other's Confidential Information without the prior written consent of the other party. Except as expressly provided in this Agreement, no ownership or license rights are granted in any Confidential Information.

8. REPRESENTATIONS AND WARRANTIES

8.1 PN. PN represents and warrants that:

(a) The person who has signed this Agreement on behalf of PN has the authority to enter into this Agreement with Microsoft on behalf of PN and to obligate PN to perform under the Agreement according to its terms;

(b) The PN Player and Upgrades thereto do not violate or infringe any copyright, trade secret, trademark, or, to the knowledge of PN, patent rights of any third party.

MSS 000638
CONFIDENTIAL

8.2 Microsoft. Microsoft represents and warrants that:

The person who has signed this Agreement on behalf of Microsoft has the authority to enter into this Agreement with PN on behalf of Microsoft and to obligate Microsoft to perform under the Agreement according to its terms.

8.3 Continuation of Representations and Warranties. The representations and warranties contained in this Section 8 are continuous in nature and shall be deemed to have been given by the parties upon the Effective Date and at each stage of performance hereunder.

9. TERMINATION

9.1 Term. This Agreement shall commence upon the Effective Date and continue in full force and effect for a period of two (2) years thereafter unless terminated earlier in accordance with the terms of this Section 9; provided, that the term shall be automatically extended for an additional period of one (1) year unless PN has, at any time during the initial two (2) year term of the Agreement, implemented modifications to ASF that Microsoft has not included in the PN Player and have been rejected by Microsoft pursuant to Section 3.1(c) or failed to exclusively use DirectDraw and DirectShow in the PN Clients beyond functionality that Microsoft has included in the PN Player for the features described in sub-sections (a)-(c) of Section 4.1. Notwithstanding the above, if both parties so agree, this Agreement may be renewed for an additional one (1) year term.

9.2 Termination By Either Party For Cause. Either party may suspend performance and/or terminate this Agreement immediately upon written notice at any time upon the occurrence of one or more of the following events:

(a) the other party is in material breach of Section 7 and fails to cure that breach within five (5) days after written notice thereof;

(b) Microsoft delivers the Rejection Notice to PN described in Section 6.1; or

(c) the other party is in material breach of any other Section of this Agreement and fails to cure such breach within thirty (30) days after written notice thereof. Any breach by PN of the obligations described in Sections 3, 4, and 6 of this Agreement shall be deemed a material breach of this Agreement.

(d) Microsoft is in material breach of Sections 5 or 6.6.

9.3 Effect of Termination.

(a) Neither party shall be liable to the other for damages of any sort resulting solely from terminating this Agreement in accordance with its terms.

(b) Any licenses already validly granted by either party as of the effective date of termination shall not be affected and shall survive termination.

9.4 Survival. In the event of expiration or termination of this Agreement for the reasons described in Sections 9.2(a), 9.2(c), or 9(d), the following Sections shall survive: Sections 3.1 (license portions only), 4.2, 6.5, 6.7, 6.8, 6.9, 7, 8, 9, 10, and 11 shall survive. In the event of termination of this Agreement for the reason described in Section 6.1, no Sections of this Agreement shall survive.

10. LIMITATION OF LIABILITIES

EXEPT FOR BREACHES OF SECTIONS ". NEITHER PARTY SHALL BE LIABLE FOR ANY INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR SPECIAL DAMAGES, EVEN IF SUCH PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

11. GENERAL

11.1 Notices. All notices and requests in connection with this Agreement shall be deemed given as of the day they are received either by messenger, delivery service, or in the United States of America mails, postage prepaid, certified or registered, return receipt requested, and addressed as follows:

To PN:	To Microsoft:
Progressive Networks, Inc. 1111 Third Avenue, Suite 2900 Seattle, WA 98101 Attention: General Counsel	Microsoft Corporation One Microsoft Way Redmond, WA 98052-6399 Attention:
Phone: (206) 674-2210	Phone: (425) 882-8080
Fax: (206) 674-2695	Fax: (425) 936-7329
	Copy to: Microsoft Corporation One Microsoft Way Redmond, WA 98052-6399 Attention: Law & Corporate Affairs
	Fax: (206) 936-7409

or to such other address as a party may designate pursuant to this notice provision.

11.2 No Joint Venture. Nothing in this Agreement shall be construed as creating an employer-employee relationship, a partnership, or a joint venture between the parties.

11.3 Governing Law. This Agreement shall be governed by the laws of the State of Washington as though entered into between Washington residents and to be performed entirely within the State of Washington, and PN consents to jurisdiction and venue in the state and federal courts sitting in King County, Washington. In any action or suit to enforce any right or remedy under this Agreement or to interpret any provision of this Agreement, the prevailing party shall be entitled to recover its costs, including reasonable attorneys' fees.

11.4 Assignment. This Agreement shall be binding upon and inure to the benefit of each party's respective successors and lawful assigns; provided, however, that PN may not assign this Agreement, in whole or in part, without the prior written approval of Microsoft. For purposes of this Agreement, an "assignment" by PN shall be deemed to include, without limitation, the following: (a) a merger of PN with another party, whether or not PN is the surviving entity; (b) any transaction or series of transactions whereby a third party acquires direct or indirect power to control the management and policies of PN, whether through the acquisition of voting securities, by contract, or otherwise; or (c) the sale of more than fifty (50%) percent of PN's assets (whether in a single transaction or series of transactions). An "assignment" shall not be deemed to include an initial public offering of PN stock.

11.5 Construction. If for any reason a court of competent jurisdiction finds any provision of this Agreement, or portion thereof, to be unenforceable, that provision of the Agreement will be enforced to

the maximum extent permissible so as to effect the intent of the parties, and the remainder of this Agreement will continue in full force and effect. Failure by either party to enforce any provision of this Agreement will not be deemed a waiver of future enforcement of that or any other provision. This Agreement has been negotiated by the parties and their respective counsel and will be interpreted fairly in accordance with its terms and without any strict construction in favor of or against either party.

11.6 **Entire Agreement.** This Agreement does not constitute an offer by either party and it shall not be effective until signed by both parties. This Agreement constitutes the entire agreement between the parties with respect to the subject matter hereof and merges all prior and contemporaneous communications, provided that, except as expressly stated herein, this Agreement does not affect or amend the License/Investment Agreement or the 1996 Agreement. It shall not be modified except by a written agreement dated subsequent to the date of this Agreement and signed on behalf of PN and Microsoft by their respective duly authorized representatives.

IN WITNESS WHEREOF, the parties have entered into this Agreement as of the Effective Date written above.

MICROSOFT CORPORATION	PROGRESSIVE NETWORKS, INC.
By: Robert Muglia	By:
Name (print): ROBERT MUGLIA	Name (print): Robert Glaser
Title: VP, SERVER APPS	Title: CEO
Date: 7/19/97	Date: 7/21/97

MS8 000641
CONFIDENTIAL

ActiveX Streaming Format (ASF) Version 1

Status of this Specification

This document is the first draft of the ASF version 1 specification. It is being created to eventually become an open specification. It may therefore be widely distributed for purposes of review and comment. All comments and input should be sent to *encf@microsoft.com*.

Abstract

The ActiveX Streaming Format (ASF) is a file format that supplies storage archival capabilities for multimedia data. It supports a wide variety of media types and contains a well-defined general-purpose extension mechanism allowing new media types to flourish.

ASF files are designed to be streamed across a network at a specific bandwidth or bit rate. ASF is independent from any data communications protocols or data communications transports.

ASF currently defines streams of audio, video, images, and script commands. These elements may be combined into a single ASF file. ASF retains all forms of media (e.g., audio and video compression) and synchronization information so that when the file is played over a network, the user sees and hears the file as the file-creator intended.

The ASF framework allows for extensibility and backward compatibility.

Document Organization

1. Introduction

This specification was developed in response to the growing need for a media-independent format for the storage of streaming multimedia content. This content may have been constructed off-line or captured in real time. ASF allows content and tool developers to work to a shared specification that supports the authoring, combining, archiving, annotation and indexing of synchronized media objects without regard to original media formats or underlying transports. Multimedia information is stored into ASF as objects. These multimedia objects include audio, video, still images, events, URLs, HTML pages, and executable programs.

ASF files are explicitly designed so that its multimedia object contents can be presented across the network as streaming media. The word "streaming" in this context refers to the "playing" of multimedia content as it is received across the network as opposed to the more traditional (and less user-friendly) approach of delaying the "playing" until the entire file has been downloaded. The content of an ASF presentation occurs with a given granularity (a sample or frame) and playback rate. Streaming implies the presentation of a frame without prior knowledge of future contexts. ASF data content is designed to be played across a network "as is".

ASF multimedia objects are synchronized with each other in terms of a timeline. These objects are targeted to specific presentation rates (in bits per second) with the actual rate selected being established by the file creator. ASF content can thus be flexibly targeted for specific network environments with distinct capacity characteristics.

ASF multimedia streams can be stored on traditional file servers, HTTP servers, or specialized media servers, and can be transmitted efficiently over a variety of different network transports. These transports include TCP/IP, RTP, specialized UDP/IP protocols, ATM, and IPX/SPX.

ASF addresses a number of important issues in multimedia stream storage, such as efficient packetization, a flexible (optional) timing model, and support for a wide range of bit rates.

The default file extension for ASF files is *.asf*.

1.1. ASF Support for Existing Media Types

ASF is appropriate for storing and combining many different kinds of streaming media information at a wide variety of data rates. It has been used to represent audio at 14.4 kilobaud, sequences of high resolution images at 28.8 kilobaud, audio/image slide shows at 28.8, animation sequences, video sequences at data rates from 150 to 1000 kilobytes/sec, MPEG-1 and MPEG-2 movies, and so forth. It also can be used to encapsulate non-image information, such as URLs, HTML pages, and program data, and to synchronize this information with audio and video.

1.2. ASF Support for New Media Types

ASF supports dynamic definition of new streaming media types and their rendering engines. Media objects and their rendering engines are defined using universally unique identifiers (UUIDs). New media types and their rendering definitions can be created dynamically and included in an ASF multimedia stream. Streaming media players built to accept ASF files can use the information in the ASF representation to identify, locate, download, securely install, and execute these new rendering engines. This allows content developers to create new kinds of media without having to create new players or media servers to support them.

1.3. Error Correction

ASF addresses the potential for data loss in the underlying transport or data storage medium. It potentially allows a variety of error correction and error concealment techniques to be employed, including the use of error correcting codes and object decomposition and distribution over multiple transport or storage blocks. At this time, however, only "N"+1 parity based correction has been deployed. The capability exists to dynamically support advanced forward error correction via UUIDs. (ASF supports both global error correction and local error correction. Global error correction could performed on a stream of packets. Local error correction could be performed on a stream or object level.)

The error correction object contains provisions for opaque information needed by the error correction engine for recovery. For example, if the error correction scheme is simple "N"+1 parity, then "N" must be available. A transformation program may also use this same information to create a new version of an ASF Multimedia stream for a server with different characteristics or using a different network transport.

1.4. Timing Model

ASF provides an optional, general timing model for use within the data streams. The model allows one to specify the definition of the clock (size, initial value, and so forth). The timing model is particularly important when synchronizing multiple disparate media types with different clock definitions. For those multimedia files in which timing information is contained in the payload (for example, MPEG), ASF clocks need not be present.

1.5. Extensibility and Version Control

Each ASF multimedia object is identified by a UUID. This identification process permits new media types to be readily supported by ASF and identified by their UUID. This flexibility permits multiple distinct versions of the "same" media type to be supported in a transparent fashion.

1.6. Protocol Independence

ASF does not contain data communications protocol dependencies that will influence which data communications protocol will carry ASF data. ASF files are similarly without operating system dependencies.

1.7. Streaming

ASF multimedia content may consist of one or more multimedia data types. Should multiple data types be supported, this data is stored within a single data object. This data object is designed to be streamed over a network (e.g., Internet or Intranet) as a unit. This optional synchronization, therefore, usually occurs via data interleaving – unless the streamed media format has its own built-in synchronization approach (e.g., MPEG).

2. Technical Descriptions

An ASF multimedia stream consists of multiple logical sections. These sections are:

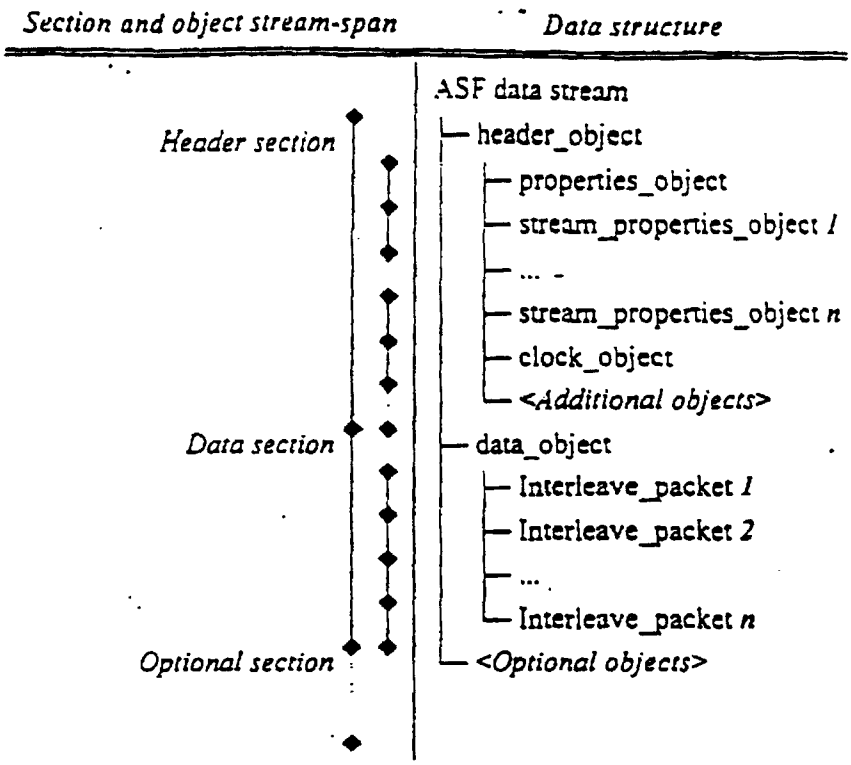
ASF v1 File Format	Required/ Optional	Number of object instances	Description
Header Section	Required	One only	The Header Section describes the ASF multimedia stream as a whole. It provides global information as well as specific information about the content contained within the stream, optional index information, optional key information, and media stream definition information. This component could be transmitted separately over a reliable protocol.
Data Section	Required	One only	This section contains the multimedia data contents represented as a linearized stream of packets.
Index Section	Optional	One only	This section contains index entries to packets within the Data Stream. The index section will not be subject to streaming, but can be used for fast lookup,search and maintenance. It can also describe important information within the multimedia content such as video key frames.
Other object	Optional	One or more	The ASF definition permits another section object, identified by its own unique UUID, to be defined.

The ASF Header Section must be the first section presented in an ASF file. The Data Section must be the second section within the file and any optional objects may then follow in any order.

The *Header Section* spans the header_object. It aggregates an array of objects with descriptive information for contents and layout of the Data Section. The *Data Section* spans the data_object. It aggregates an array of interleaved packets. The *Optional Section* spans any following optional stream components.

The ASF file is made up of several objects which may contain sub-objects. Sub-objects are objects which lack the ability to embed (or contain) other objects within themselves. The format of each object and subobject is identified by a UUID value which uniquely identifies it. The UUID is immediately followed by a 64 bit size field which gives the length of that object in bytes. These size fields delineate objects from each other and greatly enhance navigation within an ASF file.

Each of the ASF v1 file format sections listed above is composed of its own object (i.e., header object, data object, index object, other creator-defined object). The 64-bit size field of these objects includes both the length of the object itself as well as the cumulative lengths of the sub-objects which are contained in it. The size field of each of the sub-objects solely indicates the length of that sub-object itself. This relationship provides the foundation for rapid navigation within an ASF file as is graphically displayed by the following figure.



2.1. ASF Header Section

The header section is identified by the header_object whose size field specifies the length (in bytes) of the ASF Header Section. A valid header_object must contain a properties_object, a clock_object and at least one stream_properties_object

The properties_object contains the properties intrinsic to this multimedia stream, such as the UUID, overall size, and playback and transmission duration.

The stream_properties_object defines properties associated with a particular media stream. There must be a separate stream_properties_object for each media stream present in the multimedia stream (e.g., audio, video, URL flips). This object includes such information as the UUID defining that media stream, common information shared across samples, transport specific information, the error concealment strategy, and information on locating and downloading the associated rendering engines.

The clock_object defines properties for the timeline for which events are synchronized and multimedia objects are presented. This object includes such information as size, granularity, and initial value.

In addition to these required objects, the header section can also include optional objects, as shown in the following table:

Objects which can be contained by the header_object	Required/ Optional	Number of object instances	Description
properties_object	Required	One only	Describes the properties for this multimedia stream.
stream_properties_object	Required	One or more	Defines properties associated with a particular media stream
clock_object	Required	One only	Defines the clock for the playback timeline.
content_description_object	Optional	One only	Describes, in Unicode, the author, title, copyright, rating, etc.
error_correction_object	Optional	One only	Describes the algorithm needed for a particular type of error correction.
script_command_object	Optional	One only	Contains a collection of commands that each can be executed somewhere on the playback timeline.
marker_object	Optional	One only	Allows an arbitrary list of specific points on the playback timeline such as identifying the beginning of a track on a CD.
codec_object	Optional	One only	Provides a facility to embed information about the codec(s) dependencies which is needed to render the stream(s) of data.

Each object contained by the header_object is identified by a UUID. This allows for future expansion. Additional aggregated header objects can be defined as needed and identified by their own new UUIDs. Because each object starts with a UUID and a size value, media viewers that do not recognize these UUIDs can use the size value to skip the unknown object and examine the next object in the header.

The information contained in the header section must be received reliably before data streaming starts. For stored streams the header may be placed at the beginning of the multimedia stream. For broadcast streams, this information must be received out-of-band.

The format of the ASF header section is defined in the following subsections.

2.1.1. header_object

The header_object composes the ASF header section. It identifies the number of other objects which have been contained within itself (i.e., the number_headers field). A valid ASF file must have a properties_object, clock_object and at least one stream_properties_object contained within the header_object.

Data stored in the header_object itself is always little endian and byte-aligned. (This will hopefully ensure that it can always be read.) All other successive objects, like the aggregated objects of the header_object and the Data Section, are aligned and have the endian characteristics specified by the alignment and architecture fields of the header_object.

The fields of the header object occur in the following order. Appendix B defines the meaning of the 'Type' values.

Fields of the header_object	Bit size	Type	Description
object_id	128	uuid	Contains the UUID for the header_object. The header_object UUID is 0x75b22630, 0x668e, 0x11cf, 0xa6, 0xd9, 0x00, 0xaa, 0x00, 0x62, 0xce, 0x6c.
size	64	uimsbf	A number specifying the size of the entire ASF header section in bytes. Note that this value varies with the number and sizes of the enclosed objects.
number_headers	32	uimsbf	A number specifying the number of objects contained within the ASF header section. This number does not include the header_object within the count.
alignment	8	uimsbf	This field specifies the packing alignment of the following objects in the header (not the header itself and not the following data). The value 1 indicates byte alignment, 2 indicates word alignment, and so on. This is similar to the #pragma pack directive in the C language. The header is always byte aligned.
architecture	8	uimsbf	This field identifies the computer architecture type of the Data Section, Index Section (if any) and Other Object Section (if any). The value 1 indicates little endian and the value 2 indicates big-endian. (Note: The ASF Header Section itself is always little-endian.)

2.1.2. properties_object

The properties_object describes various media stream properties, such as the length of the multimedia stream, the duration, and the preferred packet size. The properties_object is separate from the header_object in order to allow its version to change (with a new UUID) without needing to redefine the header_object's format.

The properties_object refers to the global characteristics of all of the multimedia streams found within the Data Section (i.e., a grouping of the distinct streams). Each stream is then individually defined by its own stream_properties_object.

The fields of the properties_object format is given below. The differences in format of the other versions are identified in the bullets below.

Fields of the properties_object	Bit size	Type (see Appendix B)	Description
object_id	128	uuid	There are currently three versions of the properties_object. The version specified here has a unique UUID of 0x8cabdca1, 0xa947, 0x11cf, 0x8e, 0xe4, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65
size	64	uimsbf	This field identifies the size in bytes of the properties_object.
multimedia_stream_id	128	uuid	This field uniquely identifies the multimedia stream contents of this ASF file. This UUID value must match the multimedia_stream_id data element in the data_object of the Data Section.
total_size	64	uimsbf	A 64-bit quantity expressing the size, in bytes, of the entire multimedia stream. This data element is not necessarily valid when the flags broadcast bit (bit 0) is set to 1.
created	64	filetime	A 64-bit filetime indicating when the multimedia stream was created. This data element is not necessarily valid when the flags broadcast bit (bit 0) is set to 1.
num_packets	64	uimsbf	A 64-bit quantity defining the number of packets entries present in the data section. This data element is undefined when the flags broadcast bit (bit 0) is set to 1.
play_duration	64	time	A 64-bit number corresponding to the time needed to play the multimedia stream in 100-nanosecond units. This data element is not necessarily valid when the flags broadcast bit (bit 0) is set to 1. The play_duration value subtracted from the send_duration results in the clock license time that needs to be added to the clock to completely play the file. This is useful when sending non-constant bit rate files on a constant bit rate server.
send_duration	64	time	A 64-bit number corresponding to the time to send the multimedia stream in 100-nanosecond units. This data element is undefined when the flags broadcast bit (bit 0) is set to 1. The play_duration value subtracted from the send_duration results in the clock license time that needs to be added to the clock to completely play the file. This is useful when sending non-constant bit rate files on a constant bit rate server.
preroll	64	uimsbf	A 64-bit number that contains the amount of time to buffer data before starting to play. The value represents the number of 100 nanosecond clock ticks. It is usually added to a player default preroll value and used to compensate for network jitter.
flags	32	uimsbf	A 32-bit quantity representing various bit-level flags. Bit 0: broadcasting. A control flag stating whether or not the stream is being broadcast. The semantics of the total_size, created, duration and num_packets data elements are not necessarily valid if this bit is set.
min_packet_size	32	uimsbf	A 32-bit quantity expressing the size, in bytes, of the smallest packet in the data section of the multimedia stream. This data element exists to help determine if the multimedia stream is playable from servers that are constrained in their packet sizes. For streams with fixed packet sizes, the minimum and maximum packet size values are the same.
max_packet_size	32	uimsbf	A 32-bit quantity expressing the size, in bytes, of the largest packet in the data section of the multimedia stream. This data element exists to help determine if the multimedia stream is playable from servers that are constrained in their packet sizes. For streams with fixed packet sizes, the minimum and maximum packet size values are the same.
maximum_bit_rate	32	uimsbf	This 32-bit quantity contains the maximum instantaneous bit rate, in

2.1.3. stream_properties_object

The stream_properties_object describes generic media stream properties and other information that will be needed by multiple samples, such as default palettes for bitmaps or compression headers for codecs.

There must be a stream_properties_object associated with every media stream type contained in the ASF multimedia stream.

Fields of the stream_properties_object	Bit size	Type (see Appendix B)	Description
object_id	128	uuid	There are currently two versions of the stream_properties_object. The version specified here has a unique UUID of 0xb7dc0791, 0xa9b7, 0x11cf, 0x8e, 0xe6, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65.
size	64	uimsbf	A 64-bit quantity describing the size of this object in bytes.
stream_type	128	uuid	This field contains the UUID that defines the media type of the stream.
error_concealment_strategy	128	uuid	This field contains the UUID that identifies the error concealment strategy of the stream. For example, an error concealment strategy for an uncompressed bitmap stream might be to redistribute the pixels of a given bitmap across a number of packets, so that if a packet is lost sequential pixels are not lost.
offset	64	time	A 64-bit number corresponding to an offset of the stream with respect to the timeline of the program. This value is added to all of the time stamps of the samples in the stream, and can be used to indicate the presentation time of the first sample within the ASF multimedia stream. The time value represents 100-nanosecond clock ticks.
type_specific_len	32	uimsbf	A 32-bit unsigned integer identifying the number of bytes in the following type_specific_data field.
error_concealment_len	32	uimsbf	A 32-bit unsigned integer identifying the number of bytes in the error_concealment_data field.
stream_number	16	bsmbf	A 16-bit value that is used in packets as an alias to the stream properties object in order to conserve space while identifying the stream uniquely.
reserved	32	bsmbf	This field is currently not used.
type_specific_data	Array of Octets	bsmbf	A stream type has certain properties associated with it as defined by the stream type's UUID. For example, video data streams would define a well-known window size structure for this section; audio data streams might require certain header information. That data would be placed here.
error_concealment_data	Array of Octets	bsmbf	An error concealment strategy usually has certain properties associated with it, defined by the error_concealment_strategy. For example, an audio data stream might need to know how codec chunks were redistributed, or it might need a sample of encoded silence.

The type_specific_data field is an array of bytes. The length of the array (in bytes) is the value of the type_specific_len field.

The error_concealment_data field is an array of bytes. The length of the array (in bytes) is the value of the error_concealment_len field.

2.1.4. clock_object

The clock_object defines properties for the timeline for which events are synchronized and against which multimedia objects are presented. This object includes such information as size, granularity, and initial value.

The fields of the clock_object occur in the following order. Appendix B defines the meaning of the "Type" values.

Fields of the clock_object	Bit size	Type	Description
object_id	128	uuid	The UUID to identify the clock_object is 0x5fbf03b5, 0xa92e, 0x11cf, 0x8e, 0xe3, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65
size	64	uimsbf	This field identifies the size in bytes of the clock_object.
packet_clock_type	128	uuid	This field identifies the UUID of the clock_type used by this object.
packet_clock_size	16	uimsbf	This field identifies the clock size.
clock_specific_len	32	uimsbf	This field identifies the size in bytes of the clock_specific_data field
clock_specific_data	Array of Octets	bsmbf	This field contains the clock specific data of the clock identified in the packet_clock_type field.

The clock_specific_data field is an array of bytes. The length of the array (in bytes) is the value of the clock_specific_len field.

The following UUIDs have been defined for the specific clock type alternatives of the packet_clock_type field:

- CLSID_CAsfPacketClock1: 0xabd3d211, 0xa9ba, 0x11cf, 0x8e, 0xe6, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65. The granularity of this clock type is that it has a 32 bit source value and a 16 bit duration value. This is the clock type for Audio/Video Interleaved (AVI), Quicktime (MOV), and Waveform (WAV) files
- CLSID_CAsfPacketClock2: 0xabd3d213, 0xa9ba, 0x11cf, 0x8e, 0xe6, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65. The granularity of this clock type is that it has a 64 bit source value and a 32 bit duration value.
- CLSID_CAsfPacketClock3: 0xabd3d214, 0xa9ba, 0x11cf, 0x8e, 0xe6, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65. The granularity of this clock type is that it has a 64 bit source value and a 64 bit duration value.

2.1.5. content_description_object

The ASF content_description_object permits content authors to include information such as a title, copyright, author name, rating, and other description information in an ASF multimedia stream.

The fields of the content_description_object occur in the following order. Appendix B defines the meaning of the "Type" values.

Fields of the content_description_object	Bit size	Type	Description
object_id	128	uuid	The UUID for the content_description_object is 0x75b22633, 0x668e, 0x11cf, 0xa6, 0xd9, 0x00, 0xaa, 0x00, 0x62, 0xce, 0x6c
size	64	uimsbf	A 64-bit quantity describing the size of this object in bytes.
title_len	16	uimsbf	A 16-bit quantity indicating the number of Unicode characters within the title field of the multimedia stream.
author_len	16	uimsbf	A 16-bit quantity indicating the number of Unicode characters within the author field of the multimedia stream.
copyright_len	16	uimsbf	A 16-bit quantity indicating the number of Unicode characters within the copyright statement of the multimedia stream.
description_len	16	uimsbf	A 16-bit quantity indicating the number of Unicode characters within the description field of the multimedia stream.
rating_len	16	uimsbf	A 16-bit quantity indicating the number of Unicode characters within the rating field for the multimedia stream.
title	Unicode string	wchar	An array of Unicode characters that contain the title of the multimedia stream.
author	Unicode string	wchar	An array of Unicode characters that contain the name of the multimedia stream author.
copyright	Unicode string	wchar	An array of Unicode characters that contain the copyright statement of the multimedia stream.
description	Unicode string	wchar	An array of Unicode characters that contain the description of the multimedia stream.
rating	Unicode string	wchar	An array of Unicode characters that contain rating information for the multimedia stream.

The title field is an array of Unicode characters. The length of the array (in wchar) is the value of the title_len field.

The author field is an array of Unicode characters. The length of the array (in wchar) is the value of the author_len field.

The copyright field is an array of Unicode characters. The length of the array (in wchar) is the value of the copyright_len field.

The description field is an array of Unicode characters. The length of the array (in wchar) is the value of the description_len field.

The rating field is an array of Unicode characters. The length of the array (in wchar) is the value of the rating_len field.

2.1.6. error_correction_object

The error_correction_object defines the error correction method. This allows different error correction schemes to be used during content creation. The error correction object contains provisions for opaque information needed by the error correction engine for recovery. For example, if the error correction scheme is simple "N"+1 parity, then "N" must be available.

The fields of the error_correction_object occur in the following order. Appendix B defines the meaning of the "Type" values.

Fields of the error_correction_object	Bit size	Type	Description
object_id	128	uuid	Contains the UUID for this object. The error_correction_object UUID is 0x75b22635, 0x668e, 0x11cf, 0xa6, 0xd9, 0x00, 0xaa, 0x00, 0x62, 0xce, 0x6c
size	64	uimsbf	A 64-bit quantity describing the size of this object in bytes.
error_correction_id	128	uuid	A UUID defining this error correcting methodology.
error_correction_len	32	uimsbf	A 32-bit quantity indicating the number of bytes that follows in the correction_data field.
correction_data	Array of Octets	bsmbf	An array of octets which compose the opaque error correcting data for this method.

The correction_data field is an array of bytes. The length of this array (in bytes) is the value of the error_correction_len field.

2.1.7. script_command_object

Script commands can be embedded as a table in the ASF file's optional script_command_object. Script commands "ride" the ASF file to the client (e.g., the client's NetShow On-Demand Player) where they are picked up by event handlers and executed. If the receiving Web page or application does not have the correct event handlers to use the commands, the events are not handled and the playing of the ASF file plays ignores the script command.

Two distinct script command types are currently implemented:

1. *URL*. The URL command type causes the client's browser to be executed so as to display the indicated URL.
2. *Filename*. The Filename command type launches another ASF file. This may be used for such things as chained "continuous play" audio or video presentations. This command is similar to the URL command type except that its parameter specify the MMS protocol (or some other underlying data communications protocol) and an ASF file.

The fields of the script_command_object occur in the following order.

Fields of the script_command_object	Bit size	Type	Description
object_id	128	uuid	The script_command_object's UUID is 0x1efb1a30, 0xcd2, 0x11c0, 0xa3, 0x9b, 0x0, 0xa0, 0xc9, 0x3, 0x48, 0xf6
size	64	uimsbf	A 64-bit quantity describing the size of this object in bytes
command_ID	128	uuid	Identifies the structure of the command entry identified in the command_entity_struc
num_commands	16	uimsbf	Specifies the total number of script commands to be executed
num_types	16	uimsbf	This field specifies the total number of different types of script command types that have been specified. Thus, it also specifies the number of unicode strings to be found in the type_names array.
type_names	Array of type_names_struc	See Below	This is an array of type_names_struc entities. There are num_types entities in this array. The type_names field within the type_names_struc specifies a script command type name (e.g., "URL" or "filename" or "text"). The value of the type field within the command_entry structure gives the index that corresponds to that command type's Unicode-encoded name in this type_names array.
command_entry	See below	See below	This structure identifies what command should be executed at which point in the timeline.

The type_names_struc structure is formed as follows:

Fields of the type_names_struc	Bit size	Type	Description
type_names_len	16	uimsbf	This field specifies the number of Unicode characters in the type_names array.
type_names	Unicode string	wchar	This is a Unicode string whose length (in Unicode characters) is the value of type_names_len.

The Command_entry field forms the previously mentioned "table of script commands". This table is composed of one or more elements (the actual number is indicated in the num_commands field) each having the following structure:

Fields of the command_entry_structure	Bit size	Type	Description
time	32	uimsbf	This field specifies when (on the timeline) this script command is to be executed.
type	16	uimsbf	Index into the type_names array that indicates the start of a Unicode string for that command type within the array
parameter	type_names_struc	See above	The parameter value for this script command type. (E.g., an example of a parameter is "www.microsoft.com" for a URL script command type. Note: frames within a URL are indicated by a preceding "&&".) This field is a single instance of the type_names_struc structure.

The command_entry filed is an array of command_entry_structure elements. The number of entries is the value of the num_commands field.

2.1.8. marker_object

A marker is a pointer to a specific time within the ASF Data Section. Markers are not required, but they are helpful for users, especially if the contents of the data object are long. Markers enable users to quickly jump forward or backward to specific data points (e.g., audio tracks).

A "marker" thus can be viewed as a (crude) index into the data_object. Markers control the points into which the user can seek into the file contents. It is the only source of seek points for illustrated audio. (Note: video can also use the index for seeking.) Markers can be optionally used to delineate natural content divisions within the data object. For example, if the data object consisted of a series of (audio) songs then a marker could be used to delineate the beginning of each of the songs. Markers are pointers that exist to allow seeking.

The marker_object defines a set of points on the playback timeline each of which are tagged with a logical name. A marker can, for example, identify the beginning of a track on a CD or a episode in a video.

Depending on where the marker(s) is placed within the data object, the client (e.g., the Microsoft NetShow On-Demand Player) may still need to read a small portion of the ASF file preceding that marked time to load the necessary data. That is, if the indicated point requires state to have been established in order to be rendered correctly, the seek actually goes to a point previous to the marker so that the correct data will be sent at the marker time.

The fields of the marker_object occur in the following order. Appendix B defines the meaning of the "Type" values.

Fields of the marker_object	Bit size	Type	Description
object_id	128	uuid	The marker_object UUID is 0xf487cd01, 0xa951, 0x11cf, 0x8e, 0xe6, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65
size	64	uimsbf	A 64-bit quantity describing the size of this object in bytes.
marker_id	128	uuid	Contains the UUID that identifies the marker data strategy.
num_entries	32	uimsbf	Number of marker entries.
entry_alignment	16	uimsbf	Identifies byte alignment of marker data
name_len	16	uimsbf	Number of unicode characters that make up the name field.
name	Unicode string	wchar	An array of Unicode characters that contain the name of the marker object. This is the name of the entire marker object.
marker_data	See below	See below	Marker entry field. The number of marker_data_structure entries in this field is the value of the num_entries field.

The name field is an array of Unicode characters. The length of the array (in wchar) is the value of the name_len field.

The marker data is a table that is composed of the zero or more marker entries. The number of marker entries is indicated by the value of the num_entries field. Each marker entry has the following structure:

Fields of the marker_data_structure	Bit size	Type	Description
offset	64	uimsbf	Offset in bytes defines a relative distance from start of packets in the data object indicating the position of this marker entry.
time	64	uimsbf	Time of the marker entry.
entry_len	16	uimsbf	A 16-bit quantity indicating the size (in octets) of the entry_data field.
entry_data	Array of octets	bsmbf	An array of octets. The number of octets in this array is the value of the entry_len field within this structure.

The marker_data field is an array of marker_data_structure elements. The number of entries is the value of the num_entries field.

2.1.9. codec_object

The codec_object provides a mechanism to embed information about a particular codec dependency which is needed to permit the rendering of the data streams by that codec. It consists of a list of codec types (only ACM and ICM are currently implemented) and a descriptive name which enables the construction of a codec property page on the client.

The fields of the codec_object occur in the following order. Appendix B defines the meaning of the "Type" values.

Fields of the codec_object	Bit size	Type	Description
object_id	128	uuid	The UUID to identify the codec_object is 0x86d15240, 0x311d, 0x11d0, 0xa3, 0xa4, 0x0, 0xa0, 0xc9, 0x3, 0x48, 0xf6
size	64	uimsbf	This field identifies the size in bytes of the codec_object.
codec_ID	128	uuid	This field identifies the UUID of the codec_type used by this object.
codec_entry _len	32	uimsbf	This field identifies the number of CodecEntry entries in the codec_entry field
codec_entry	See below	See below	This field contains the codec specific data that is an array of CodecEntry elements.

The codec_entry field is an array of CodecEntry structures. The numcer of entries in this array is the value of the codec_entry_len field.

The CodecEntry structure is defined as follows:

Fields of the CodecEntry structure	Bit size	Type	Description
type	16	uimsbf	This field identifies the type of codec it is. The current values have been defined to date: ICM_CODEC 0x0001 ACM_CODEC 0x0002 UNKNOWN_CODEC 0xffff
name_len	16	uimsbf	The number of Unicode characters in the name field.
name	Array of Unicode	wchar	Name of the codec in Unicode characters.
description_len	16	uimsbf	The number of Unicode characters in the description field.
description	Array of Unicode	wchar	Description of the codec in Unicode characters.
cbinfo_len	16	uimsbf	Number of bytes in the cbinfo field.
cbinfo	Array of octets	bsmbf	Array of octets identifying the type specification of that codec.

2.2. ASF Data Section

The data portion of an ASF multimedia stream is a packetized representation of media stream samples. ASF admits many possible interleaving strategies, allowing a multimedia stream to be optimized for a particular network.

The data section is composed of a series of data packets. The mechanism for transmitting these packets is beyond the scope of this document. De-multiplexing of the stream may occur at either

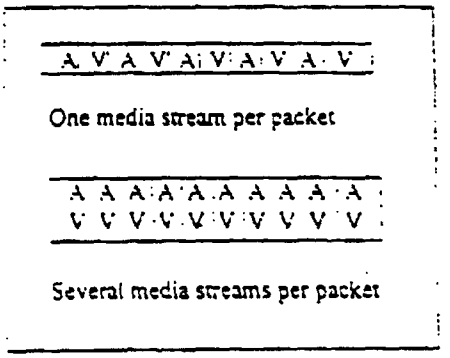
the transmission or rendering end-station, or at any intermediate node (such as a mixer or translator). The packet definition is transport independent and provides mechanisms to support transmission over reliable and unreliable protocols. The data packet was designed with both flexibility and efficiency in mind. Capabilities provided within the packet definition include:

- Single or multiple payload types per packet
- Fixed or variable sized packets.
- Error correction information. (optional)
- Clock information. (optional)
- Redundant sample information (for example, presentation time stamp). (optional)
- Sequence number. (optional)

The data section of the format consists of the data_object and one or more packets.

The data_object marks the beginning of the ASF data section and is used to correlate the header and data sections of an ASF media stream. The multimedia_stream_id data element in the data_object must match the multimedia_stream_id data element in the properties_object.

The heart of the ASF multimedia stream is the packet structure (see Section 2.2.2). A packet structure contains one or more payloads of data. Each packet may contain the data from a single media stream or interleaved data from several media streams. For example, a multimedia stream which is comprised of both an audio stream and a video stream may be packetized in either of the following ways:



Each payload in a packet (see Section 2.2.2) can contain a stream_id data element. This value corresponds to the stream_number data element of the stream_properties_object.

2.2.1. data_object

The data_object is header information that exists to uniquely define the ASF Data Section and to correlate the ASF Header Section with the ASF Data Sections within an ASF Media stream (i.e., an ASF file).

The data_object must be the first element within the ASF Data Section – it must always precede the first packet.

In the broadcast case, this object should be received reliably when reading the headers. In this case the num_packets and size field is not valid.

The fields of the data_object occur in the following order:

Fields of the data_object of the ASF Data Section	Bit size	Type	Description
object_id	128	uuid	Contains the UUID for this object. The marker_object UUID is 0x75b22636, 0x668e, 0x11cf, 0xa6, 0xd9, 0x00, 0xaa, 0x00, 0x62, 0xce, 0x6c
size	64	uimsbf	A 64-bit quantity describing the size of the ASF Data Section in bytes (i.e., both the data_object and the packet).
multimedia_stream_id	128	uuid	A UUID uniquely identifying an ASF multimedia stream. In order to have a valid ASF multimedia stream the multimedia_stream_id field in the data_object must match the multimedia_stream_id field in the properties_object.
num_packets	64	uimsbf	A 64-bit quantity defining the number of packets present in the data section.
alignment	8	uimsbf	Specifies the packing alignment within packets. The value 1 indicates byte alignment, 2 indicates word alignment and so on. This is similar to the #pragma pack directive in the C language and is expected to be usually 1.
packet_alignment	8	uimsbf	Specifies the packet packing alignment. For example, if a file were generated for a server which required 128 byte alignment on network packets (for DMA addressing) this value would be 128.

2.2.2 packet

The packet represents the heart of the ASF multimedia stream. It is here that the actual multimedia data contents are stored as packets. A "packet" is a collection of multimedia data which is ready to be streamed "as is" over the Internet/intranet. Ideally the packet has been correctly sized so that all that needs to be done to ship it "over the wire" is to append the appropriate data communication protocol headers. The minimum packet size is 512 bytes. The maximum is (data communications) protocol dependent but is generally less than 56K bytes.

The packet data must immediately follow the data_object within the ASF Data Section. Except when broadcast is true, the num_packets field within the data_object (see Section 2.2.1) indicates how many distinct packets are contained within the ASF Data Section. For example, if the value of the num_packets field is "70" then there will be seventy occurrences of the packet data within that file's ASF Data Section. Each of these packet instances are a distinct data packet.

Each of these data packets themselves may be interleaved (i.e., composed of data from multiple multimedia streams.) Each multimedia stream is identified by the value of the stream_id field within the payload_structure of the packet. This value correlates to the appropriate steam_number field value within stream_properties_object of the ASF header (see Section 2.1.3). The number of interleaved elements within a data packet is identified by the number_payloads value within the payload_flag field of the packet. The data of each interleaved element is contained within an instance of the payload_structure, which makes up the payload_struc field of the packet.

The format of the packet data is quite complex in order to ensure that the packet data is as dense as possible for efficient transmission over a network. This section presents the packet's format in a presentation approach that is consistent with the rest of the specification.

An alternative presentation of this format is contained within Appendix A. The presentation in Appendix A uses the ISO/IEC MPEG standard presentation approach which should be intuitive for people familiar with the C programming language.

The syntax of the packet varies depending upon the value of the most significant bit of its initial byte (i.e., the error_correction_bit). The fields of the packet occur in the following order:

Fields of the packet of the ASF Data Section	Bit size	Type	Description
initial_structure	See below	See below	The format of the initial structure varies depending upon the value of the error_correction_bit.
stream_flag	8	bsmbf	The byte consists of four two-bit fields: The two most significant bits are the **stream_id_type** value. These two bits indicate the size of stream_id field: '00' 0 bits (No stream_id data present) '01' 8 bits '10' 16 bits '11' Illegal value The second most-significant two bits are the **object_id_type**. This field indicates the number of bits of object ID data present. The following values are defined: '00' 0 bits (No object_id data present) '01' 8 bits '10' 16 bits '11' 32 bits The third to the most significant two bits are the **offset_type** bit field which indicates the number of bits of offset data present: '00' 0 bits(No offset data present) '01' 8 bits '10' 16 bits '11' 32 bits The least significant two bits are the **replicated_data_type** field which indicates the number of bits for the replicated_data_len element: '00' 0 bits(No replicated_data_len data present) '01' 8 bits '10' 16 bits '11' 32 bits
packet_len	0, 8, 16, or 32	uimsbf	(Optional field) Contains a number that indicates the packet length size. The size of this field is based upon the value specified by packet_len_type within the flag field.
sequence	0, 8, 16, or 32	uimsbf	(Optional field) Contains a sequence number for the packet. The size of this field is determined by the value specified by the sequence_type within the flag field.
padding_len	0, 8, 16, or 32	uimsbf	(Optional field) Contains a number that specifies the number of padding bytes present at the end of the packet (i.e., padding field). The size of this field is determined by the value specified by the padding_len_type within the flag field.
clock_data	48, 96, or 128	uimsbf	Contains the data representing time information (i.e., the clock's source and duration field values). This information is presented in the packet_clock_type format (see Section 2.1.4). The version of the packet_clock_type specifies the exact format and length of this field.
payload_flag	8	uimsbf	This flag byte contains two fields. The most significant two bits make up the **payload_len_type**. This value indicates the number of bits present in the payload_len field:

			'00' 0 bits – payload_len field not existent '01' 8 bits '10' 16 bits '11' 32 bits The six least significant bits makes up the **number_payloads**. This 6-bit unsigned integer value specifies the number of payload packets are present (i.e., the number of payload_structures in the payload_struc array).
payload_struc	See Below	See Below	This field contains the payload information: a packet's data. This field is an array of payload_structure. Each element within this array potentially represents a distinct interleaved element within the data packet.
padding	Array of Bytes	bsmbf	The 'data' bytes of the packet padding.

The padding field is an array of bytes. The length of this array (in bytes) is the value of the padding_len field.

The format of the initial_structure varies depending upon the most significant bit of the first byte (i.e., the error_correction_present bit). If that bit is cleared (i.e., is '0') then the initial structure consists of the following byte:

Fields of the initial_structure when the error_correction_present bit is cleared	Bit size	Type	Description
flag	8	bsmbf	This byte consists of five fields given in descending order from most significant bit to least significant bit. **error_correction_present** bit is the most significant bit. When set to 1, this packet is participating in error correction information and contains information specific to the particular error correction method. **packet_len_type** is made up of two bits which indicate the size of the packet_len field: '00' 0 (No packet_len data present) '01' 8 bits '10' 16 bits '11' 32 bits The **padding_len_type** consists of two bits which indicate the size of the padding_len field. '00' 0 (No padding_len data present) '01' 8 bits '10' 16 bits '11' 32 bits The **sequence_type** consists of two bits which indicate the size of the sequence field: '00' 0 (No sequence data present) '01' 8 bits '10' 16 bits '11' 32 bits The **multiple_payloads_present** bit is the least significant bit. To allow efficient transmission of media stream samples, we must allow for the concatenation of parts of multiple samples into a single packet. When this bit has the value of 1, there will be data from multiple media stream samples in the packet.

The most significant bit of the first byte of the initial_structure is the error_correction_present bit. If it is set (equal to 1) then the initial_structure consists of the following fields:

Fields of the initial_structure when the error_correction_present bit is set	Bit size	Type	Description
ec_flag	8	csmbf	This byte consists of four fields presented in descending order from most significant bit. The **error_correction_present** bit is the most significant bit. When set to 1, this packet is participating in error correction information and contains information specific to the particular error correction method. The **error_correction_len_type** is a two bit field which indicate the size of the error_correction_data_len field: '00' last 4 bits within this byte contains the length info and error_correction_data_len field does not exist. '01' 8 bits '10' 16 bits '11' 32 bits The **opaque_data** bit indicates whether opaque data exists or not. The **error_correction_data_length** is a four-bit field. If the error_correction_len_type has the value of '00' then these four bits contain the error_correction_data_len value and the error_correction_data_len field (next field below) does not exist. In this case this four-bit value determines the length of the error_correction_data array in bytes. Otherwise the value of this field is '0000'.
error_correction_data_len	0, 8, 16, or 32	bmsbf	(Optional field) This field specifies the number of bytes in the error_correction_data array. The size of this field is determined by the value of the error_correction_len_type above. This field does not exist if error_correction_len_type's value is '00'.
error_correction_data	Array of bytes	csmbf	The actual per-packet data required to implement the selected error_correction_method. The number of bytes in this array is specified by the value of the error_correction_data_len field if it exists. If it doesn't exist then the array length is specified by the value of error_correction_data_length.
opaque	Array of bytes	csmbf	(Optional field) If the opaque_data bit in the ec_flag field is set (equal to '1') then this is an opaque packet and this field exists, otherwise it does not. If this field exists then the remainder of the fields in the packet structure is ignored and the remainder of the data in packet is opaque data.
flag	8	csmbf	The byte consists of five fields given in order from most significant bit to least significant bit. The **reserved** bit is the most significant bit. It is set to '0'. The **packet_len_type** is made up of two bits which indicate the size of the packet_len field:

			'00' 0 (No packet_len data present) '01' 8 bits '10' 16 bits '11' 32 bits The **padding_len_type** consists of two bits which indicate the size of the padding_len field: '00' 0 (No padding_len data present) '01' 8 bits '10' 16 bits '11' 32 bits The **sequence_type** consists of two bits which indicate the size of the sequence field: '00' 0 (No sequence data present) '01' 8 bits '10' 16 bits '11' 32 bits The **multiple_payloads_present** bit is the least significant bit. To allow efficient transmission of media stream samples, we must allow for the concatenation of parts of multiple samples into a single packet. When this bit has the value of 1, there will be data from multiple media stream samples in the packet.

The **payload_struc** is an array that is composed of the one or more payload_structure entries. The number of payload_structure entries is computed as follows: If the multiple_payloads_present bit (within the flag field) is cleared (equal to '0') then there is only one instance of the payload_structure. If it is set (equal to '1') then the value of the number_payloads field (within the payload_flag field) determines the number of payload_structure entries within the table(i.e., the table in this case is an array of payload_structures). Each payload_structure entry has the following format:

Fields of the payload_structure	Bit size	Type	Description
stream_id	0, 8, 16, or 32	bmsbf	(Optional field) Identifies the stream type of the payload. The stream_id corresponds to the stream_number field of a stream_properties_object (in the ASF Header Section: see Section 2.1.3) of this media stream. The length of the stream_id field depends on the value of stream_id_type within the stream_flag field.
object_id	0, 8, 16, or 32	bmsbf	(Optional field) Identifies the object identifier. The length of the object_id field depends on the value of object_id_type within the stream_flag field.
offset	0, 8, 16, or 32	bmsbf	(Optional field) The length of the offset field depends on the value of offset_type value within the stream_flag field. The significance of this field is that each packet payload is part or all of a media stream sample. We must be able to recombine payloads into a whole media stream sample, so we need to know the bytes within the final object contained in this payload. A double, {offset, length}, is required for each payload. The offset represents the starting address within the zero-address-based media stream sample where the packet payload should be copied. The length of this field is found either explicitly in the payload_len data element (in the case of multiple payloads; see below) or implicitly from the packet size (in the case of a single payload).
replicated_data_len	0, 8, 16, or	bmsbf	(Optional field) This field specifies the number of bytes of replicated data present in the replicated_data element below

	32		The size (and existence) of this field is determined by the value of replicated_data_type within the stream_flag field.
replicated_data	Array of byte	bmsbf	(Optional field) Contains the data bytes of the opaque replicated data. Replicating all sample properties makes error concealment possible allowing continued operation in an environment with an unreliable data source. The replicated_data_len field gives the length (in bytes) of this array
payload_len	0, 8, 16, or 32	bmsbf	(Optional field) This field specifies the number of payload bytes present in the current payload that is the number of bytes in the payload array. The size (and existence) of this field is determined by the value of payload_len_type.
payload_data	Array of byte	bmsbf	The data bytes of the opaque media stream sample data in the packet payload. The array size (in bytes) is calculated as follows: When multiple_payloads_present is set to '1', payload_len is used as the value. When multiple_payloads_present has the value 0, the payload data must be calculated from the overall packet size.

The payload_struc field is an array of one or more payload_structure elements. The number of entries is the value of the number_payloads field if the multiple_payloads_present is set otherwise there is only one payload_structure element.

The replicated_data field is an array of bytes. The length of this array (in bytes) is the value of the replicated_data_len field.

2.3. ASF Index Section

An index_object describes index information associated with the multimedia stream. Index information is commonly used for video files to point to key frames. This facilitates both fast forward and fast rewinds. Fast forwards and fast rewinds up to 10x are supported within ASF. Both time-related indexes and byte-offset indexes can be defined.

The fields of the index_object occur in the following order. Appendix B defines the meaning of the "Type" values.

Fields of the index_object of the ASF Index Section	Bit size	Type	Description
object_id	128	uuid	The UUID for the index_object is 0x33000890, 0xe5b1, 0x11cf, 0x89, 0xf4, 0x0, 0xa0, 0xc9, 0x3, 0x49, 0xcb.
size	64	uimsbf	A 64-bit quantity describing the size of the index_object in bytes.
index_id	128	uuid	Contains a UUID that uniquely identifies this index type. The type can be used to group related entries, such as entries that use a common data type in the entry_data data element.
time_delta	64	uimsbf	Time interval between index entries.
max_packets	32	uimsbf	Maximum value for packet_count
num_entries	32	uimsbf	A 32-bit unsigned integer describing the number of index entries that are defined within the index_info array.
index_info	See below	See below	An array of the index_information structure. This contains the index information.

index_info is an array that is composed of one or more instances of the index_information structure. The number of entries within the array is indicated by the value of the num_entries field. Each index_information has the following structure.

Fields of the index_information	Bit size	Type	Description
packet	32	uimsbf	Packet number associated with this index entry
packet_count	16	uimsbf	Number of packet to send with this index entry

Appendix A Alternative packet Description

The following is an alternative way to describe the packet format of Section 2.2.2. The format described in this section is identical to the format described in Section 2.2.2. Only the format description approach is different. This approach uses the bit stream notation of the ISO/IEC MPEG standards. It should also be familiar to people who know the C programming language.

```
packet() {
error_correction_present                                    1 bsmbf
if (error_correction_present==1) {
   error_correction_len_type                                2 bsmbf
   opaque_data                                              1 bsmbf
   if (error_correction_len_type=='00') {
      error_correction_data_len                             4 bsmbf
   } else {
      reserved                                              4 '0000'
      if (error_correction_len_type=='01') {
         error_correction_data_len                          8 bsmbf
      } else if (error_correction_len_type=='10') {
         error_correction_data_len                         16 bsmbf
      } else if (error_correction_len_type=='11') {
         error_correction_data_len                         32 bsmbf
      }
   }
   for (i = 0; i < error_correction_data_len; i++) {
      error_correction_data                                 8 bsmbf
   }

   if (opaque_data) {
      for (i = 0; i < inferred_opaque_len; i++) {
         opaque_data_contents                               8 uimsbf

   }                          // end of if (opaque_data)
                               // end if error_correction_present
}                             // not in orig; put here cause section ends
if ( ! opaque_data) {         // with opaque data if
                              // error_correction_present// opaque data
                              is ignored if err corr not pres

   packet_len_type                                          2 bsmbf
   padding_len_type                                         2 bsmbf
   sequence_type                                            2 bsmbf
   multiple_payloads_present                                1 bsmbf
   stream_id_type                                           2 bsmbf
   object_id_type                                           2 bsmbf
   offset_type                                              2 bsmbf
   replicated_data_type                                     2 bsmbf
   if (packet_len_type != '00') {
      if (packet_len_type == '01') {
         packet_len                                         8 uimsbf
      } else if (packet_len_type == '10') {
         packet_len                                        16 uimsbf
      } else if (packet_len_type == '11') {
         packet_len                                        32 uimsbf
      }
```

```
                                    end if packet_len_type != '00'
if (sequence_type != '00') {
   if (sequence_type == '01') {
      sequence                                                  8 uimsbf
   } else if (sequence_type == '10') {
      sequence                                                 16 uimsbf
   } else if (sequence_type == '11') {
      sequence                                                 32 uimsbf
   }
}                           // end if sequence_present != '00'
if (padding_len_type != '00') {
   if (padding_len_type == '01') {
      padding_len                                               8 uimsbf
   } else if (padding_len_type == '10') {
     padding_len                                               16 uimsbf
   } else if (padding_len_type == '11') {
     padding_len                                               32 uimsbf
   }
}                           // end if padding_len_type != '00'
for (i = 0; i < inferred_clock_len; i++) {
   clock_data                                                   8 uimsbf
}
if (multiple_payloads_present == 1) {
   payload_len_type                                             2 uimsbf
   number_payloads                                              6 uimsbf
}                           // end if multiple_payloads_present
do {                        // for each packet
   if (stream_id_type != '00') {
      if (stream_id_type == '01') {
         stream_id                                              8 bsmbf
      } else if (stream_id_type == '10') {
         stream_id                                             16 bsmbf
      } else if (stream_id_type == '11') {
         stream_id                                             32 bsmbf
      }
   }                                // end if stream_id_type != '00'
   if (object_id_type != '00') {
      if (object_id_type == '01') {
         object_id                                              8 bsmbf
      } else if (object_id_type == '10') {
         object_id                                             16 bsmbf
      } else if (object_id_type == '11') {
         object_id                                             32 bsmbf
      }
   }                                // end if object_id_type != '00'
   if (offset_type != '00') {
      if (offset_type == '01') {
         offset                                                 8 bsmbf
      } else if (offset_type == '10') {
         offset                                                16 bsmbf
      } else if (offset_type == '11') {
         offset                                                32 bsmbf
      }
   }                                // end if offset_type != '00'
   if (replicated_data_type != '00') {
      if (replicated_data_type == '01') {
         replicated_data_len                                    8 bsmbf
      } else if (replicated_data_type == '10') {
```

```
            replicated_data_len                                    16 bsmbf
         } else if (replicated_data_type == '11') {
            replicated_data_len                                    32 bsmbf
         }
         for (i=0; i< replicated_data_len; ++i) {
            replicated_data                                        8 bsmbf
         }
      }                          // end if replicated_data_type != '00'
      if (multiple_payloads_present) {
         if (payload_len_type == '01') {
            payload_len                                            8 uimsbf
         } else if (payload_len_type == '10') {
            payload_len                                            16 uimsbf
         } else if (payload_len_type == '11') {
            payload_len                                            32 uimsbf
         }
      }                          // end if multiple_payloads_present
      for (i=0; i < payload_len_or_calc_len; ++i) {
         payload_data                                              8 bsmbf
      }
   } while (multiplePayloads && i++ < number_payloads);
}                               // end of opaque != '1'
if (padding_len_type != '00') {
   for (i=0; i< padding_len; ++i) {
      padding                                                      8 bsmbf
   }
}                               // end if padding_len_type != '00'
}                               // end if (! opaque_data)
}                               // end packet
```

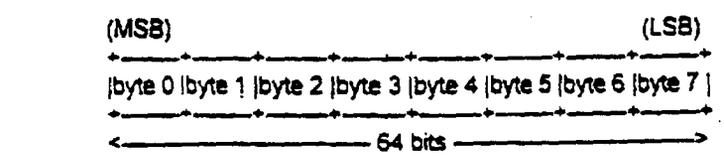
Appendix B Bit Stream Types

The bit stream type describes the target data type and the order of transmission of bits in the coded bit-stream. The bit stream types are 'smbf, filetime, uuid, time, uimsbf, and wchar.

bsmbf
Bit string, most significant bit first. Bit strings are written as a string of ones and zeros within single quote marks, for example '1000 0001.' Blanks within a bit string are for ease of reading and have no other significance.

filetime

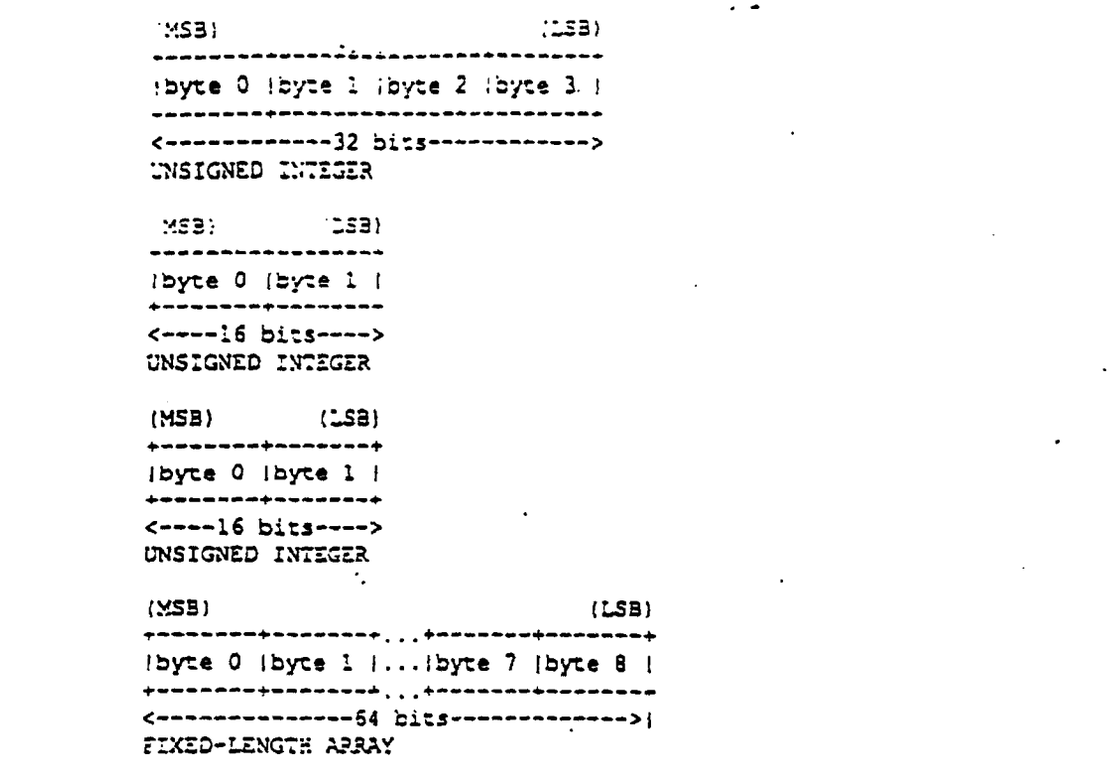
A 64-bit integer that contains a time stamp corresponding to the number of 100 nanosecond ticks since January 1, 1601. The following diagram demonstrates the filetime format:



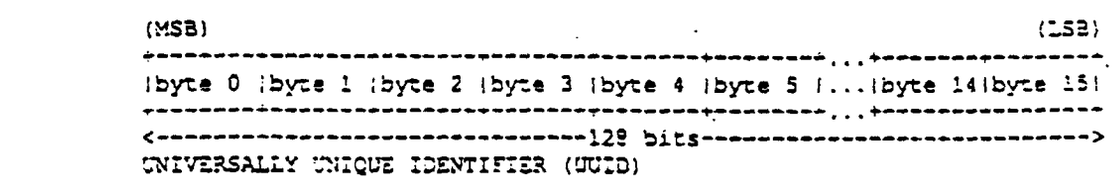
MS8 000668
CONFIDENTIAL

uuid

The universally unique identifier is a 128 bit (16 octet) data structure composed of a 32 bit unsigned integer, two 16-bit unsigned integers, and an array of eight octets. The constituent parts are shown in the following diagrams:



These components are concatenated to form the UUID:



time

64-bit unsigned integer that represents a count of 100-nanosecond clock ticks. Note that time values represent relative values that can be associated with any starting point, in contrast to the absolute value represented by the filetime that is defined to have a fixed starting point.

uimsbf

Unsigned integer, most significant bit first. This implies that if the integer is a multiple of octets, for example a WORD, the integer is in big-endian format.

wchar
Wide character; 16 bits of Unicode character data.

Appendix C Header UUIDs

Each ASF object contains its own unique identifier. The following table lists the identifiers for currently defined ASF objects:

Note: the UUID value is given in hex.

header_object 0x75b22630, 0x668e, 0x11cf, 0xa6, 0xd9, 0x00, 0xaa, 0x00, 0x62, 0xce, 0x6c

properties_object 0x8cabdca1, 0xa947, 0x11cf, 0x8e, 0xe4, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65

stream_properties_object 0xb7dc0791, 0xa9b7, 0x11cf, 0x8e, 0xe6, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65

content_description_object 0x75b22633, 0x668e, 0x11cf, 0xa6, 0xd9, 0x00, 0xaa, 0x00, 0x62, 0xce, 0x6c

marker_object 0xf487cd01, 0xa951, 0x11cf, 0x8e, 0xe6, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65

index_object 0x33000890, 0xe5b1, 0x11cf, 0x89, 0xf4, 0x0, 0xa0, 0xc9, 0x3, 0x49, 0xcb

error_correction_object 0x75b22635, 0x668e, 0x11cf, 0xa6, 0xd9, 0x00, 0xaa, 0x00, 0x62, 0xce, 0x6c

data_object 0x75b22636, 0x668e, 0x11cf, 0xa6, 0xd9, 0x00, 0xaa, 0x00, 0x62, 0xce, 0x6c

frames_object 0x75b22637, 0x668e, 0x11cf, 0xa6, 0xd9, 0x00, 0xaa, 0x00, 0x62, 0xce, 0x6c

clock_object 0x5fbf03b5, 0xa92e, 0x11cf, 0x8e, 0xe3, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65

stream_routing_object 0x6ba83691, 0xb8d7, 0x11cf, 0x96, 0xc, 0x0, 0xa0, 0xc9, 0xa, 0x8e, 0x34

script_command_object 0x1efb1a30, 0xb62, 0x11d0, 0xa3, 0x9b, 0x0, 0xa0, 0xc9, 0x3, 0x48, 0xf6

codec_object 0x86d15240, 0x311d, 0x11d0, 0xa3, 0xa4, 0x0, 0xa0, 0xc9, 0x3, 0x48, 0xf6

Clock types:

- CLSID_CasfNullClockType: 0x1d70b760, 0x748e, 0x11cf, 0x9c, 0x0f, 0x00, 0xa0, 0xc9, 0x03, 0x49, 0xcb
- CLSID_CAsfPacketClock1 0xabd3d211, 0xa9ba, 0x11cf, 0x8e, 0xe6, 0x0, 0xc0, 0xc, 0x20, 0x53, 0xb5
- CLSID_CAsfPacketClock2 0xabd3d213, 0xa9ba, 0x11cf, 0x8e, 0xe6, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65
- CLSID_CAsfPacketClock3 0xabd3d214, 0xa9ba, 0x11cf, 0x8e, 0xe6, 0x0, 0xc0, 0xc, 0x20, 0x53, 0x65

Appendix D Glossary

Big-endian A byte-ordering scheme whereby byte 0 is the high order byte.

End-station An application that generates the content to be sent and/or consumes the content of received packets.

Little-endian A byte-ordering scheme whereby byte 0 is the low order byte.

Media Stream A single stream of data from the same media type (for example, video or audio). A media stream can be combined with other media streams to form a multimedia stream.

Multimedia Stream A combination of multiple, synchronized media streams that may be of differing media types.

Packet A collection of multimedia data ready to be streamed over the Internet/intranet. Ideally the packet has been correctly sized so that all that needs to be done is to append the data communication protocol headers to ship it "over the wire". The minimum packet size is 512 bytes. The maximum is protocol dependent and less than 56K bytes. A rule of thumb is to size it to the (bit-rate / 80).

Sample An entity within a media stream that that has an associated presentation time: a bitmap, a JPEG image, a video frame, an HTML page, and so forth. A sample is the smallest significant multimedia element which make up that particular stream type.

Transport Address The combination of a network address and a port identifying a transport-level endpoint (for example an IP address and a UDP port). Packets are transmitted from a source transport address to a destination transport address.