harmonic

Media API

RELEASE 7.2

Programmer Guide

Manual Part No. 28-0064

January 23, 2013

Disclaimer

Harmonic reserves the right to alter the equipment specifications and descriptions in this publication without prior notice. No part of this publication shall be deemed to be part of any contract or warranty unless specifically incorporated by reference into such contract or warranty. The information contained herein is merely descriptive in nature, and does not constitute a binding offer for sale of the product described herein. Harmonic assumes no responsibility or liability arising from the use of the products described herein, except as expressly agreed to in writing by Harmonic. The use and purchase of this product do not convey a license under any patent rights, copyrights, trademark rights, or any intellectual property rights of Harmonic. Nothing hereunder constitutes a representation or warranty that using any products in the manner described herein will not infringe any patents of third parties.

Trademark Acknowledgments

Harmonic and all Harmonic product names are trademarks of Harmonic Inc. All other trademarks are the property of their respective owners.

The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of those agreements.

May be covered by one or more of U.S. Patents No. 6,571,351; 6,696,996; 6,545,721; 6,574,225; 6,895,003; 6,522,649; 6,643,702; foreign counterparts and pending patent applications.

This system is distributed with certain other software that may require disclosure or distribution of licenses, copyright notices, conditions of use, disclaimers and/or other matter. Use of this system or otherwise fulfilling their conditions constitutes your acceptance of them, as necessary. Copies of such licenses, notices, conditions, disclaimers and/or other matter are available in any one of the following locations: the LEGAL NOTICES AND LICENSES directory of the distribution disk of the software, the root directory of the hard disk drive of the Products, or by contacting us at support@harmonicinc.com.

Notice

Information contained in this guide is subject to change without notice or obligation. While every effort has been made to ensure that the information is accurate as of the publication date, Harmonic Inc. assumes no liability for errors or omissions. In addition, Harmonic Inc. assumes no responsibility for damages resulting from the use of this guide.

License Agreement and Limited Warranty

1. AGREEMENT: This is a legal agreement ("Agreement") between you ("you" or "your") and Harmonic, or its appropriate local affiliate ("Harmonic", "we", "us" or "our"). Use of our product(s) and any updates thereto purchased or validly obtained by you (the "Products"), and/or the Software (as defined below) (collectively, the "System"), constitutes your acceptance of this Agreement. "Use" includes opening or breaking the seal on the packet containing this Agreement, installing or downloading the Software as defined below or using the Software preloaded or embedded in your System. As used herein, the term "Software" means the Harmonic owned software and/or firmware used in or with the Products and embedded into, provided with or loaded onto the Products in object code format, but does not include, and this Agreement does not address, any third-party or free or open source software separately licensed to you ("Third Party Software"). If you do not agree to this Agreement, you shall promptly return the System with a dated receipt to the seller for a full refund.

2. LICENSE: Subject to the terms and conditions of this Agreement (including payment), we hereby grant you a nonexclusive, nontransferable license to use the object code version of the Software embedded into, provided solely for use with or loaded onto the Product, and the accompanying documentation ("Documentation") for your internal business purposes. The Software and any authorized copies are owned by us or our suppliers, and are protected by law, including without limitation the copyright laws and treaties of the U.S.A. and other countries. Evaluation versions of the Software may be subject to a time-limited license key.

3. RESTRICTIONS: You (and your employees and contractors) shall not attempt to reverse engineer, disassemble, modify, translate, create derivative works of, rent, lease (including use on a timesharing, applications service provider, service bureau or similar basis), loan, distribute, sublicense or otherwise transfer the System, in whole or part except to the extent otherwise permitted by law. The Software may be operated on a network only if and as permitted by its Documentation. You may make one (1) back up copy of the object code of the Software for archival purposes only. Evaluation Software will be run in a lab, nonproductive environment. Results of any benchmark or other performance tests may not be disclosed to any third party without our prior written consent. Title to and ownership of the Software and Documentation, and all copyright, patent, trade secret, trademark, and other intellectual property rights in the System, shall remain our or our licensors' property. You shall not remove or alter any copyright or other proprietary rights notice on the System. We reserve all rights not expressly granted.

4. LIMITED WARRANTY: (a) Limited Warranty. We warrant to you that, commencing on your receipt of a Product and terminating 1 year thereafter, the System will perform substantially in accordance with its then-current appropriate Documentation. The Product (including replacements) may consist of new, used or previously-installed components. (b) Remedies. If the System fails to comply with such warranty during such period, as your sole remedy, you must return the same in compliance with our product return policy, and we shall, at our option, repair or replace the System, provide a workaround, or refund the fees you paid. Replacement Systems are warranted for the original System's remaining warranty period. (c) Exclusions. EVALUATION SOFTWWARE IS LICENSED ON AS-IS BASIS AND SUBJECT TO 4(d). We will have no obligation under this limited warranty due to: (i) negligence, misuse or abuse of the System, such as unusual physical or electrical stress, misuse or accidents; (ii) use of the System other than in accordance with the Documentation; (iii) modifications, alterations or repairs to the System made by a party other than us or our representative; (iv) the combination, operation or use of the System with equipment, devices, software or data not supplied by us; (v) any third party hardware or Third Party Software, whether or not provided by us; (vi) any failure other than by us to comply with handling, operating, environmental, storage or maintenance requirements for the System in the Documentation, including, without limitation, temperature or humidity ranges. (d) Disclaimers. We are not responsible for your software, firmware, information, or data contained in, stored on, or integrated with any Product returned to us for repair or replacement. SUCH

LIMITED WARRANTY IS IN LIEU OF, AND WE SPECIFICALLY DISCLAIM, ANY AND ALL OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF SATISFACTORY QUALITY, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. WE DO NOT WARRANT THAT THE SYSTEM WILL MEET YOUR REQUIREMENTS OR BE UNINTERRUPTED OR ERROR-FREE. NO ADVICE OR INFORMATION, WHETHER ORAL OR WRITTEN, OBTAINED FROM US OR ELSEWHERE, WILL CREATE ANY WARRANTY NOT EXPRESSLY STATED IN THIS AGREEMENT. Some jurisdictions do not allow the exclusion of implied warranties or limitations on how long an implied warranty may last, so such exclusions may not apply to you. In that event, such implied warranties or limitations are limited to 60 days from the date you purchased the System or the shortest period permitted by applicable law, if longer. This warranty gives you specific legal rights and you may have other rights which vary from state to state or country to country.

5. LIMITATION OF LIABILITY: WE AND OUR AFFILIATES, SUPPLIERS, LICENSORS, OR SALES CHANNELS ("REPRESENTATIVES") SHALL NOT BE LIABLE TO YOU FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, OR EXEMPLARY DAMAGES OF ANY KIND, INCLUDING BUT NOT LIMITED TO LOST REVENUES, PROFITS OR SAVINGS, OR THE COST OF SUBSTITUTE GOODS, HOWEVER CAUSED, UNDER CONTRACT, TORT, BREACH OF WARRANTY, NEGLIGENCE, OR OTHERWISE, EVEN IF WE WERE ADVISED OF THE POSSIBILITY OF SUCH LOSS OR DAMAGES. NOTWITHSTANDING ANY OTHER PROVISIONS OF THIS AGREEMENT, WE AND OUR REPRESENTATIVES' TOTAL LIABILITY TO YOU ARISING FROM OR RELATING TO THIS AGREEMENT OR THE SYSTEM SHALL BE LIMITED TO THE TOTAL PAYMENTS TO US UNDER THIS AGREEMENT FOR THE SYSTEM. THE FOREGOING LIMITATIONS SHALL NOT APPLY TO DEATH OR PERSONAL INJURY TO PERSONS OR TANGIBLE PROPERTY IN ANY JURISDICTION WHERE APPLICABLE LAW PROHIBITS SUCH LIMITATION. YOU ARE SOLELY RESPONSIBLE FOR BACKING UP YOUR DATA AND FILES, AND HEREBY RELEASE US AND OUR REPRESENTATIVES FROM ANY LIABILITY OR DAMAGES DUE TO THE LOSS OF ANY SUCH DATA OR FILES. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO SUCH EXCLUSIONS MAY NOT APPLY TO YOU.

6. CONFIDENTIALITY: Information in the System and the associated media, as well as the structure, organization and code of the Software, are proprietary to us and contain valuable trade secrets developed or acquired at great expense to us or our suppliers. You shall not disclose to others or utilize any such information except as expressly provided herein, except for information (i) lawfully received by the user from a third party which is not subject to confidentiality obligations; (ii) generally available to the public without breach of this Agreement; (iii) lawfully known to the user prior to its receipt of the System; or (iv) required by law to be disclosed.

7. SUPPORT: Updates, upgrades, fixes, maintenance or support for the System (an "Upgrade") after the limited warranty period may be available at separate terms and fees from us. Any Upgrades shall be subject to this Agreement, except for additional or inconsistent terms we specify. Upgrades do not extend the limited warranty period.

8. TERM; TERMINATION: The term of this Agreement shall continue unless terminated in accordance with this Section. We may terminate this Agreement at any time upon default by you of the license provisions of this Agreement, or any other material default by you of this Agreement not cured with thirty (30) days after written notice thereof. You may terminate this Agreement any time by terminating use of the System. Except for the first sentence of Section 2 ("License") and for Section 4(a) ("Limited Warranty"), all provisions of this Agreement shall survive termination of this Agreement. Upon any such termination, you shall certify in writing such termination and non-use to us.

9. EXPORT CONTROL: You agree that the Products and Software will not be shipped, transferred, or exported into any country or used in any manner prohibited by the United States Export Administration Act or any other export laws, restrictions, or regulations (the "Export Laws"). You will indemnify, defend and hold us harmless from any and all claims arising therefrom or relating thereto. In addition, if the Products or Software are identified as export controlled items under the Export Laws, you represent and warrant that you are not a citizen, or otherwise located within, an embargoed nation (including without limitation Iran, Iraq, Syria, Sudan, Libya, Cuba, North Korea, and Serbia) and that you are not otherwise prohibited under the Export Laws from receiving the Software. All rights to the Products and Software are granted on condition that such rights are forfeited if you fail to comply with the terms of this Agreement.

10. U.S. GOVERNMENT RIGHTS: The Software and the documentation which accompanies the Software are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government as end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Harmonic, 4300 North First Street, San Jose, CA 95134 U.S.A.

11. GENERAL: You shall not assign, delegate or sublicense your rights or obligations under this Agreement, by operation of law or otherwise, without our prior written consent, and any attempt without such consent shall be void. Subject to the preceding sentence, this Agreement binds and benefits permitted successors and assigns. This Agreement is governed by California law, without regard to its conflicts of law principles. The U.N. Convention on Contracts for the International Sale of Goods is disclaimed. If any claim arises out of this Agreement, the parties hereby submit to the exclusive jurisdiction and venue of the federal and state courts located in Santa Clara County, California. In addition to any other rights or remedies, we shall be entitled to injunctive and other equitable relief, without posting bond or other security, to prevent any material breach of this Agreement. We may change the terms, conditions and pricing relating to the future licensing of our Systems and other intellectual property rights, including this Agreement, from time to time. No waiver will be implied from conduct or failure to enforce rights nor effective unless in a writing signed on behalf of the party against whom the waiver is asserted. If any part of this Agreement is found unenforceable, the remaining parts will be enforced to the maximum extent permitted. There are no third-party beneficiaries to this Agreement. We are not bound by additional and/or conflicting provisions in any order, acceptance, or other correspondence unless we expressly agree in writing. This Agreement is the complete and exclusive statement of agreement between the parties as to its subject matter and supersedes all proposals or prior agreements, verbal or written, advertising, representations or communications concerning the System.

Every reasonable attempt has been made to comply with all licensing requirements for all components used in the system. Any oversight is unintentional and will be remedied if brought to the attention of Harmonic at support@harmonicinc.com.

Documentation Conventions

This guide may use some special symbols and fonts to call your attention to important information. The following symbols appear throughout this guide:

DANGER: The Danger symbol calls your attention to information that, if ignored, can cause physical harm to you.

CAUTION: The Caution symbol calls your attention to information that, if ignored, can adversely affect the performance of your Harmonic product, or that can make a procedure needlessly difficult.

LASER DANGER: The Laser symbol and the Danger alert call your attention to information about the lasers in this product that, if ignored, can cause physical harm to you.

NOTE: The Note symbol calls your attention to additional information that you will benefit from heeding. It may be used to call attention to an especially important piece of information you need, or it may provide additional information that applies in only some carefully delineated circumstances.

IMPORTANT: The Important symbol calls your attention to information that should stand out when you are reading product details and procedural information.

TIP: The Tip symbol calls your attention to parenthetical information that is not necessary for performing a given procedure, but which, if followed, might make the procedure or its subsequent steps easier, smoother, or more efficient.

In addition to these symbols, this guide may use the following text conventions:

Convention	Explanation
`Typed Command`	Indicates the text that you type in at the keyboard prompt.
<Ctrl>, <Ctrl>+<Shift>	A key or key sequence to press.
Links	The *italics in blue* text to indicate Cross-references, and hyperlinked cross-references in online documents.
Bold	Indicates a button to click, or a menu item to select.
`ScreenOutput`	The text that is displayed on a computer screen.
Emphasis	The *italics* text used for emphasis and document references.

NOTE: You require Adobe Reader or Adobe Acrobat version 6.0 or later to open the PDF files. You can download Adobe Reader free of charge from www.adobe.com.

Contents

Introduction 1
About the Media API 1
What is Essence? 1
What is a Media File? 1
What is a Movie File? 1
Required Environment for Using API 2
Files included in the API Package 2
New in this Version 3
Spectrum System Documentation Suite 3
Technical Support 5
Useful Information when Contacting Technical Support 5

Chapter 1: Media API Functions and Parameters 7
Understanding the Media API 7
Structure of API 7
Create New Movies Class (OmMediaCopier) 7
Functions of OmMediaCopier Class 8
Query Movie Properties Class (OmMediaQuery) 26
Functions of the OmMediaQuery Class 26
Media Property Class (OmMediaProperty) 46
Functions of the Media Property Class 47
Media Property List Class (OmMediaPropertyList) 50
Functions of the Media Property List Class 50
Software Version Class (struct OmSwVersion) 53
Functions of the Omneon Software Version Class 53
Data Types 56

Appendix A: Glossary 77
Abbreviations and Terms Used in Omneon Media API 77

Appendix B: Notes about the Media API 78
About File Access 78
About File and Wrapper Formats used by Spectrum 79
Media API Support for and Integration with Third Party Applications 80
Limitations on References to Material Files and Jumps in QuickTime Files 80
Notes on Parsing QuickTime Files 80
Constructing QuickTime Movies from Scratch for Export to an Spectrum MediaDirector 81
Notes on Renaming Files in Media API 81

Appendix C: Omneon Neutral VANC Format 83
About Omneon Neutral VANC Format 83
Frame Header Format 83

Omneon Frame Header Type-1 83
Omneon Frame Header Type-2 84
Omneon VANC Packet 86
Omneon VANC Packet Header 86
Modified VANC Payload 87
VANC Usage Notes 88

Appendix D: Contacting the Technical Assistance Center 89

Introduction

The Spectrum™ System supports a Media API written in C++ that you can use to write applications to implement file format parsers or use generic APIs to query or modify file format files.

This document is intended for third-party application integrators who want to copy or query media files. If you need to understand how to play media files, refer to the *Player API Guide* for more information.

The document consists of the following:

- *Introduction* outlines the contents of the guide, provides customer service information, and lists new additions or changes to this version of the Media API.
- *Media API Functions and Parameters* provides descriptions and examples of Media classes, members, and parameters.

In addition, supplemental information is packaged as follows:

- *Glossary* provides a list of abbreviation and terms used in this document and in the Media API files. A list of supported media types is also included.
- *Notes about the Media API* provides answers to some of the most frequently asked questions relating to the Media API.
- *Omneon Neutral VANC Format* provides information about the neutral vertical ancillary data (VANC) format.

About the Media API

This function set is the File Format Application Programming Interface, (from here on referred to as the Media API). This section introduces the concepts of Essence, Media File, and Movie File.

What is Essence?

Essence is the raw video, audio, and data streams.

What is a Media File?

A media file is a file that contains one or more video or audio tracks of data. Media files contain the actual footage (also called content or essence). This API treats both audio and video media in an identical manner, so that while one file may contain audio and video, another might contain just audio.

What is a Movie File?

A Movie File usually refers to a file composed of both audio and video. It is frequently associated with Apple's QuickTime files and has the file extension ".mov." A Movie File may be self-contained (i.e contain the essence within it), or it may contain reference to external essence. Spectrum MediaDirectors typically use referenced files when capturing media.

NOTE: Refer to *Appendix A, GlossaryA* for descriptions of additional concepts and terms referred to in the Media API.

Required Environment for Using API

To co-exist with the API files, your application must be built in the following environment:

- Built to run on a PC that is using one of the following operating systems:
  - Microsoft Windows XP, Windows 2008 Service Pack 1, or Windows 7
  - Linux Fedora Core 4 or later, 32-bit (other distributions are likely to work)
  - Linux Fedora Core 4 or later, 64-bit (other distributions are likely to work)
- Your code and your compiler must be able to co-exist with C++ code. The supplied header files include classes; they also use such items as the C++ approach of automatically assigning "typedefs" for each structure and enumeration.

NOTE: This guide supports API development with the Microsoft® C++ compilers included with Visual Studio® 6 and later on Windows, and with the GCC compiler version 4.0 or later on Linux. Other compilers and languages are likely to work, however, the developer must write the interface code necessary to load the library and invoke its functions.

- This API assumes that the size of "int" is 32 bits. Enumerated values are also sized at 32, "bool" is sized at 8 bits. Pointers are 32 bits in Windows and Linux-32, and 64 bits on Linux-64.
- Have a structure alignment of 8.

Files included in the API Package

This version of the Media API includes the following files:

- The "include" directory holds the following four files:
  - omdefs.h
  - ommedia.h
  - ommediadefs.h
  - omtcdata.h

These files are needed for compiling and can be installed in a location such as with other external libraries.

- The "lib" directory holds:
  - ommedia.lib (Windows) or libommedia.a (Linux)

This file is needed for linking and can be installed in a location such as with other external libraries.

- The "bin" directory holds:
  - ommedia.dll

This file is needed for running your application and should be installed in a Windows System Directory or the Omneon System Program directory.

  - OmQtChooser.exe (Windows) or OmQtChooser (Linux)
  - OmMediaDebugLog.exe (Windows) or OmMediaDebugLog (Linux)

These programs are used to configure the API for the type of QuickTime wrapper and for logging debug information respectively. They are not wrapped for running your application.

- The "src" directory holds:
  - Three filenames with a .cpp extension
  - Four filenames with a .dsp extension (Windows only)
  - One filename with a .dsw extension (Windows only)

- One filename with a .Makefile extension (Linux only)

These files contain sample code.

- The “doc” directory holds:
  - Readme.txt:

This file provides last minute information regarding this version of the ommedia.dll.

  - MediaAPI_version#_ Guide.pdf

(This document in pdf format.)

New in this Version

There is no new information in this release.

Spectrum System Documentation Suite

The table below describes the documents which comprise the Spectrum System Documentation Suite.

.

Spectrum (MediaDirector 2201, 2202, MediaCenter, MediaPort 5000, MediaPort 7000, ChannelPort)	
This document...	**Provides this information...**
Spectrum System Installation and Hardware Reference Guide	■ System installation ■ Software installation and upgrade details ■ Orientation to system components including MediaDirectors, MediaCenters, MediaPorts, ChannelPorts, and MediaStores ■ Troubleshooting system components ■ Specifications for system components
Spectrum System Protocol Reference Guide	■ Command sets and preroll parameters for controlling MediaDirectors ■ The Omneon implementation of FTP server
Spectrum Quick Reference Guides	■ Front and back panel views of Spectrum devices ■ LED assignments and legends
Spectrum ChannelPort Template Authoring Guide	ChannelPort template authoring
Spectrum ChannelPort Tools User Guide	Using ChannelPort tools
Spectrum Onboard Playout Control and PlayoutTool User Guide	■ Onboard Playout Control Overview ■ Using PlayoutTool
Spectrum Component Replacement Guide	Component replacement instructions for all Spectrum devices
Spectrum and MediaDeck Release Notes	Last minute information regarding a product release
Spectrum MediaDeck 7000	
This document...	**Provides this information...**

Spectrum MediaDeck 7000 User Guide	■ System installation ■ Upgrade instructions ■ Orientation to system components
Spectrum MediaDeck 7000 Installation Guide	System installation
Spectrum and MediaDeck Release Notes	Last minute information regarding a product release
Spectrum MediaDeck 7000 Read Me First	■ Passwords for downloading MediaDeck and SystemManager files ■ Instructions for obtaining and installing the license file for SystemManager ■ Installation overview
Spectrum (MediaDirector 2100, 2101, 2102, 2102B, MediaPort 1000, 3000, 4000, 6000 Series)	
This document...	**Provides this information...**
Spectrum System Getting Started Guide	■ System installation ■ Software installation and upgrade details
Spectrum System Hardware Orientation Guide	■ Orientation to system components including MediaPorts, and MediaStores ■ Troubleshooting system components ■ Specifications for system components
Spectrum System Protocol Reference Guide	■ Command sets and preroll parameters for controlling MediaDirectors ■ The Omneon implementation of FTP server
Spectrum Quick Reference Guides	■ Front and back panel views of Spectrum devices ■ LED assignments and legends
Omneon MediaDeck	
This document...	**Provides this information...**
Omneon MediaDeck User Guide	■ System installation ■ Upgrade instructions ■ Orientation to system components
Omneon MediaDeck Installation Guide	System installation
Spectrum and MediaDeck Release Notes	Last minute information regarding a product release
Omneon MediaDeck Read Me First	■ Passwords for downloading MediaDeck and SystemManager files ■ Instructions for obtaining and installing the license file for SystemManager ■ Installation overview

All items are packaged in self-extracting files and available for download from the Omneon FTP site at the following location: *ftp://ftp.omneon.com//updates/omneon/Current/Spectrum/*

- **Release Notes**: Spectrum_v7.2.0.0_ReleaseNotes.pdf
- **All other components of the Spectrum System Documentation Suite**: MediaDeckAndSpectrum-v7.2.0.0-Documentation.exe

- **For ChannelPort templates, tools, and documentation, as well as Onboard Playout Control tools and documentation**: ChannelPortTemplatesAndTools-7.2.0.0-SWandDoc.exe.

For Spectrum media and wrapper formats, refer to: Spectrum_Media_and_WrapperFormats.pdf in the Spectrum System Documentation Suite.

For the SystemManager documentation, navigate to:

ftp://ftp.omneon.com/updates/omneon/current/SystemManager

SystemManager documents are packaged **SystemManager-v5.25.0.0-Documentation.exe**.

All files on the Omneon Support Server are password protected. Contact Harmonic *Technical Support* if you need assistance with unlocking the files.

Technical Support

For information on contacting Harmonic Technical Support, refer to *Appendix D, Contacting the Technical Assistance Center*.

Useful Information when Contacting Technical Support

In order to assist Harmonic Technical Support, review the following information:

- **What version of firmware is installed on your system?**

From the **Home** tab, click the **Upgrade Firmware** icon in the left-hand column to display the **Upgrade Firmware** page. The firmware version for each device is shown in the **Current Firmware Version** column.

- **What version of SystemManager software is installed?**

From SystemManager, click the **Help** tab. The version is shown in the **Server Software** section of the page.

- **Which Windows operating system is running on the SystemManager client PC?**
  a. From Windows, click the **Start** button, and then click **Run**.
  b. In the **Open** field, type: winver, and then press **Enter** to open the **About Windows** dialog box, which shows the version number.
- **How much memory is installed on the SystemManager platform? (for example, 256 MB, 512 MB, or 1 GB)**
  a. From Windows, click the **Start** button, and then click **Run**.
  b. In the **Open** field, type: winver and then press **Enter** to open the **About Windows** dialog box. Look for the line which reads "Physical memory available to Windows."
- **Please provide the manager.oda file from the SystemManager platform or client PC**

Harmonic Technical Support may request that you email the manager.oda file, which contains configuration information for your system. This file is located on the SystemManager platform at D:\Omneon\Manager\omdb, or if you are using a client PC with a single C: partition, it will be in the same directory on the C: drive.

- **What is the model and serial number of the hardware involved?**
  - For Spectrum and MediaDeck devices: from the **Home** tab, click the **Upgrade Firmware** icon in the left-hand column to display the **Upgrade Firmware** page. Both MediaDirectors and MediaDecks are listed in the **MediaDirectors** section. Find the Model Numbers and Serial Numbers listed in their respective columns.

Scroll down to the **MediaPorts** section to view the Model Numbers and Serial Numbers for MediaPorts and MediaDeck Modules.

- For Omneon MediaGrid Devices: Click the **Servers & Switches** icon in the left-hand column. From the Servers and Switches page, in the **Name** column, click the link for the Omneon MediaGrid device to open the **Properties** page for that device.
- For ProXchange devices: Click the ProXchange Servers icon in the left-hand column. From the **Servers** page, in the **Name** column, click the link for the ProXchange device to open the **Properties** page for that device.
- For ProBrowse devices: Click the ProBrowse Servers icon in the left-hand column. From the **Servers** page, in the **Name** column, click the link for the ProBrowse device to open the **Properties** page for that device.
- For MAS devices: Click the MAS Servers icon in the left-hand column. From the Servers page, in the **Name** column, click the link for the MAS device to open the **Properties** page for that device.

For Spectrum Systems

- **What is the name of the Player that is being used?**

From SystemManager, click the **Player Configuration** link in the left-hand column, and then click the name of the MediaDirector or MediaDeck. The **Player List** page for that device appears. The names and status of all players are listed.

- **What file format and bit rate is the Player configured for? (for example, MPEG, DV, IMX?)**
  a. From SystemManager, click the **Player Configuration** link in the left-hand column, and then click the name of the MediaDirector or MediaDeck. The **Player List** page for that device appears.
  b. From the player list, click the **Properties** link to view all the details for a player.
- **If the problem is related to Ingest or Playout of a clip, what is the Clip ID involved?**

The clip name or clip ID should be indicated by whatever software application you are using to play or record video. For Omneon ClipTool, clip names are displayed in the clip management area of the ClipTool main window.

- **What brand of Automation, if any, is being used for control?**
- **Is the Automation using VDCP or API for communication control?**
- **What other third party device (for example, Tandberg* or Snell and Wilcox*) is involved?**
- Please supply the log files

Chapter 1
Media API Functions and Parameters

This section provides the following information:

- *Understanding the Media API*
- *Create New Movies Class (OmMediaCopier)*
- *Query Movie Properties Class (OmMediaQuery)*
- *Media Property Class (OmMediaProperty)*
- *Media Property List Class (OmMediaPropertyList)*
- *Software Version Class (struct OmSwVersion)*
- *Data Types*

Understanding the Media API

This document describes a *simple* API intended for customers who wish to:

- Implement file format parsers.
- Use generic APIs to query or modify the Media files.

This API is intended for Third Party Application Providers who want to copy or query media files. Refer to these sources if you need to understand the following:

- To play media files, refer to the *Spectrum Player API Guide*.
- To view media files such as DV25, DVCPro 25, AIFF, WAV, or ITU Rec601, refer to Apple's QuickTime documentation.
- To edit media files such as DV25 and DV50, refer to Apple's Final Cut Pro documentation.

Structure of API

The Media API consists of a single windows DLL (*ommedia.dll*) or a Linux static library (*libommedia.a*). There are four C++ classes:

- *Create New Movies Class (OmMediaCopier)*
- *Query Movie Properties Class (OmMediaQuery)*
- *Media Property Class (OmMediaProperty)*
- *Media Property List Class (OmMediaPropertyList)*

In addition:

- *Data Types*

  These include structures and enumerators in the interfaces as input and/or output arguments.

Create New Movies Class (OmMediaCopier)

Description

This class is used to gather media essence and create a new movie file.

Remarks

This class assumes that all essence files are synchronized at the left edge, thus they begin at the same time. When **setTrimmedAudio** has been called, if audio essence is longer than the video, the created movie will ignore the extra audio samples so that all video will have corresponding audio, and vice versa. The basic steps to create a movie are:

1. Create a new *OmMediaCopier()* instance.
2. Add essence with **add SourceTrack**(s) or **appendTrack**(s).
3. Specify an output movie with **setDestination**.
4. Perform the copy.

Functions of OmMediaCopier Class

The following are functions of the OmMediaCopier Class:

- *OmMediaCopier()*
- *~OmMediaCopier()*
- *static bool setDebug(const char * fname)*
- *bool addSourceTrack(const wchar_t* path, uint track)*

bool addSourceTrack(const wchar_t path, uint track)*

- *bool addSourceTracks(const char * path)*

bool addSourceTracks(const wchar_t path)*

- *bool appendTrack(uint dstTrack, uint srcTrack, const char * mediapath, uint inSrcFram = 0, uint outSrcFrame = ~0)*

bool appendTrack(uint dstTrack, uint srcTrack, const wchar_t mediapath, uint inSrcFram = 0, uint outSrcFrame = ~0*

- *bool appendTracks(uint dstTrack, const char * path, uint inFrame = 0, uint outFrame = ~0)*

boolappendTracks(uint dstTrack, const wchar _t path, uint inFrame = 0, uint outFrame = ~0)*

- *bool setDestination(const char * path, bool replace = false)*

bool setDestination(const wchar_t path, bool replace = false)*

- *void setCopyType(OmMediaCopyType)*
- *bool setRange(int inFrame, uint outFrame = ~0)*
- *bool setTrackRange(uint trackNum, uint inFrame, uint outFrame = ~0)*
- *void setTrimmedAudio(bool = true)*
- *bool setOutputSuffix(OmMediaFileType, char * suffix)*

bool setOutputSuffix(OmMediaFileType, wchar_t suffix)*

- *const char * getOutputSuffix(OmMediaFileType) const*

const wchar_t getOutputSuffix(OmMediaFileType) const*

- *OmMediaFileType get OutputType(const char * suffix)*

OmMediaFileType get OutputType(const wchar_t suffix)*

- *bool setRemoteHost(const char *host)*
- *bool initializeRemoteHost ()*
- *bool getDstInfo(OmMediaInfo&)*
- *bool get DstTrackInfo(uint trackNum, OmMediaSummary&)*
- *bool set MaxRecordAge(uint seconds)*
- *bool copy()*
- *OmCopyProgress getProgress()*
- *void release()*

- *void cancel()*
- *void setClipProperties (const char *clipProperties)*

OmMediaCopier()

Description

Constructs a new *OmMediaCopier*.

Parameters

None.

Return value

No values returned.

Remarks

None to note.

Limitations

None to note.

~OmMediaCopier()

Description

Destructor for *OmMediaCopier*.

Parameters

None.

Return value

No values returned.

Remarks

Multiple *OmMediaCopier* instances may exist concurrently. As each one may consume substantial resources, ensure that each instance does not leak.

Limitations

None to note.

static bool setDebug(const char * fname)

Description

Turns debug logging on or off.

Parameters

Parameter	Description
const char * fname	A text string containing a pointer to the file name of the file to open.

Return value

No values returned.

Remarks

If fname is specified, debugging will be logged to the file, thus overwriting any previous content. If fname is NULL or " ", debug logging is turned off. Note that this function logs *all* calls, not just those of OmMediaCopier.

Limitations

None to note.

bool addSourceTrack(const char * path, uint track)

bool addSourceTrack(const wchar_t* path, uint track)

Description

Adds the specified track from the specified file to the output.

Parameters

Parameter	Description
const char * path **const wchar_t* path**	A text string containing a pointer to the name of the file to open.
uint track	Specifies the track from the input file to be copied to the next unused track in the output file. Tracks are zero based.

Return value

Value	Description
bool	Returns true on success, or false if the media file is not understood or could not be opened.

Remarks

None to note.

Limitations

None to note.

bool addSourceTracks(const char * path)

bool addSourceTracks(const wchar_t* path)

Description

Adds all of the individual media essence found in the specified file to the output. For example, if the source file is DV, a video and audio track is added to the output.

Parameters

Parameter	Description
const char * path **const wchar_t* path**	A text string containing a pointer to the name of the file to open.

Return value

Value	Description
bool	Returns **true** on success, or **false** if the media file is not understood or could not be opened.

Remarks

DV files and other video file types can exist without embedded audio. In this case, only a video track is added to the output when this function is called.

If the track is shorter than the overall length of the final movie, silence or black frames (depending on track type) will be appended to the track to fill it out.

Limitations

None to note.

bool appendTrack(uint dstTrack, uint srcTrack, const char * mediapath, uint inSrcFram = 0, uint outSrcFrame = ~0)

bool appendTrack(uint dstTrack, uint srcTrack, const wchar_t* mediapath, uint inSrcFram = 0, uint outSrcFrame = ~0

Description

Appends track #srcTrack from the specified file to track #dstTrack in the output.

Parameters

Parameter	Description
uint dstTrack	A numeric value that specifies the track number in the output file to which this clip is to be appended.
uint srcTrack	A numeric value that specifies the track number in the source file from which the frames are to be copied.
const char * mediapath **const wchar_*t mediapath**	A text string containing a pointer to the name of the file to open.
uint inSrcFrame = 0	A numeric value that specifies the first frame to be copied from the source track.
uint outSrcFrame = ~0	A numeric value that specifies the first frame number in the source track which is not to be copied, e.g. frames inFrame to outFrame -1 will be copied.

Return value

Value	Description
bool	Returns **true** on success, or **false** if the media file is not understood or could not be opened, or if the track does not exist.

Remarks

If the inFrame and outFrame are specified, only the frames starting with inFrame and ending with outFrame - 1 will be appended. Note that the source track must match the type (video or audio) of the existing data in the track.

Limitations

Copying a complete MPEG Long GOP track will fail using *bool appendTrack*. Use **addSourceTrack** instead.

bool appendTracks(uint dstTrack, const char * path, uint inFrame = 0, uint outFrame = ~0)

boolappendTracks(uint dstTrack, const wchar _t* path, uint inFrame = 0, uint outFrame = ~0)

Description

Appends all tracks from the specified file to the output, starting at track #dstTrack.

Parameters

Parameter	Description
uint dstTrack	A numeric value that specifies the track number in the output file to which this clip is to be appended.
const char * path **const wchar_t* path**	A text string containing a pointer to the name of the file to open.
uint inFrame = 0	A numeric value that specifies the first frame to be copied from the source track.
uint outFrame = ~0	A numeric value that specifies the first frame number in the source track which is not to be copied, i.e. frames inFrame to outFrame -1 will be copied.

Return value

Value	Description
bool	Returns **true** on success, or **false** if the media file is not understood or could not be opened, or if the track does not exist.

Remarks

If the inFrame and outFrame are specified, only the frames starting with inFrame and ending with outFrame - 1 will be appended. Note that each of the source tracks must match the type (video or audio) of the existing data in each of the destination tracks.

bool setDestination(const char * path, bool replace = false)

bool setDestination(const wchar_t* path, bool replace = false)

Description

Creates a new movie at the specified path.

Parameters

Parameter	Description
const char * path **const wchar_t* path**	A text string containing a pointer to the name of the file to open.
bool replace = false	Specifies that if the path already exists, the path and all files referenced by the path will be deleted prior to copying. Note that if the replace is false, no files will be overwritten.

Return value

Value	Description
bool	Returns **true** on success, or **false** if the media file is not understood or could not be opened, or if the track does not exist.

Remarks

The type of file created is implied by the suffix. Possible types that can be created include .mov, .wav, .aiff, and .dv. If you specify a container movie like QuickTime (with the suffix .mov), then it is possible to copy any media essence into it. If replace is true, then the destination movie file and any referenced media files will be removed, if they exist.

Starting with release 4.6, when you call setCopyType, output is as follows:

- With OmReferenceCopy or OmFlattenedWithDiscreteMedia and an .mxf file, output is MXF op1b.
- With OmFlattenedWithEmbeddedMedia and an .mxf file, output is MXF op1a.
- With OmReferenceCopy or OmFlattenedWithDiscreteMedia and a .mov file, output is a reference movie.
- With OmFlattenedWithEmbeddedMedia and a .mov file, output is a QuickTime self-contained movie.

DV files and other video file types can exist without embedded audio. In this case, only a video track is added to the output when this function is called.

Limitations

If *MediaCopier* is used to make a reference movie (refer to **setCopyType**), rerunning the same code a second time with replace = true will delete the referenced files as well.

void setCopyType(OmMediaCopyType)

Description

Specifies the type of copy to be performed.

Parameters

Parameter	Description
OmMediaCopyType	Specifies the type of copy to be performed.

Return value

No return value.

Remarks

The default is *omFlattenedWithDiscreetMedia*.

Limitations

None to note.

See also

Refer to **OmMediaCopyType** for types of copies supported.

bool setRange(int inFrame, uint outFrame = ~0)

Description

Sets the copy range on the source(s).

Parameters

Parameter	Description
int inFrame	A numeric value that specifies the first frame to be copied from the source track.
uint outFrame = ~0	A numeric value that specifies the first frame number in the source track which is not to be copied i.e. frames inFrame to outFrame -1 will be copied.

Return value

Value	Description
bool	Implies the destination will be outFrame-inFrame frames in length. Returns **false** if outFrame <= inFrame. If outFrame is ~0, it is shorthand for the end of the input. Also, if outFrame is beyond the end of the input it, returns **false**.

Remarks

OmMediaCopier's frame numbers are always zero-based. The default range is 0,~0.

Limitations

None to note.

bool setTrackRange(uint trackNum, uint inFrame, uint outFrame = ~0)

Description

Sets the copy range on the specified track.

Parameters

Parameter	Description
uint trackNum	A numeric value that specifies the track number in the specified destination file. Tracks are zero based.
uint inFrame	A numeric value that specifies the first frame to be copied from the source track.
uint outFrame = ~0	A numeric value that specifies the first frame number in the source track which is not to be copied, i.e. frames inFrame to outFrame -1 will be copied.

Return value

Value	Description
bool	Returns **true** on success, or **false** if the media file is not understood or could not be opened.

Remarks

Track numbers begin at zero and increase with each call to **addSourceTracks**. If you are unsure about the track numbers, use **getDstInfo** and **getDstTrackInfo** to clarify.

This method can be used to specify a shift in a track relative to other tracks, or to select a subset of a source track for inclusion in the destination. For example, a call of *setTrackRange*(2,5,~0) will extract frame 5 through the end of track 2; a call of *setTrackRange*(2, -20, 10) will extract frame 0 through frame 9 from the source and place it at frame 20 through 29 in the destination.

Limitations

None to note.

See also

addSourceTracks, *getDstInfo*, **getDstTrackInfo**

void setTrimmedAudio(bool = true)

Description

Specifies that, independent of any specified outFrame, the copy should terminate on the last frame of video.

Parameters

Parameter	Description
bool = true	Specifies that audio tracks should be trimmed to match the end of video.

Return value

No values are returned.

Remarks

This is a simple way of making sure a too-long audio track does not make the final movie too long. If this occurs, the QuickTime player and a MediaDirector shows white and black frames, respectively, when there is no video frame opposite an audio frame. By default, the value is false.

Limitations

None to note.

bool setOutputSuffix(OmMediaFileType, char * suffix)

bool setOutputSuffix(OmMediaFileType, wchar_t* suffix)

Description

Specifies the extension to the file name to be used.

Parameters

Parameter	Description
OmMediaFileType	Specifies the file type.
char * suffix **wchar_t* suffix**	Specifies the path name suffix to be used. For example “m2v”.

Return value

Value	Description
bool	Returns **true** if the suffix chosen is one from the table that follows. Returns **false** if the suffix chosen is not a recognized suffix.

Remarks

The following table lists the recognized suffixes by file type.

Recognized Suffixes	File Type
dv	DV
dif	
dnxhd	DNxHD
mov	QT
mxf	MXF
mts	MPEG TS
mps	MPEG PS
mp4	MPEG 4
m2v	MPEG V
mpg	
mp2	MPEG A2
mp3	MPEG A3
hdcam	HDCAM
601	Rec 601
aiff	AIFF
aifc	
aif	
wav	WAVE
wave	

Recognized Suffixes	File Type
ac3	AC3
vbi	VBI
data	DATA
null	NULL

Limitations

None to note.

const char * getOutputSuffix(OmMediaFileType) const

const wchar_t* getOutputSuffix(OmMediaFileType) const

Description

Returns the current suffix which is being used by the file type.

Parameters

Parameter	Description
OmMediaFileType	Specifies the file type.

Return value

Value	Description
const char * const wchar_t*	Returns the suffix to be used.

Remarks

None to note.

Limitations

None to note.

See also

setOutputSuffix

OmMediaFileType get OutputType(const char * suffix)

OmMediaFileType get OutputType(const wchar_t* suffix)

Description

Returns the file type referred to by the suffix.

Parameters

Parameter	Description
const char * suffix **const wchar_t* suffix**	A text string containing a pointer to the suffix to be used.

Return value

Value	Description
OmMediaFileType	Returns the file type used. Returns **unknown** if the file type is invalid.

Remarks

None to note.

Limitations

None to note.

bool setRemoteHost(const char *host)

Description

Specifies a remote transfer managed by an external host.

Parameters

Parameter	Description
const char *host	Specifies the host name or IP address of the transfer agent. The following systems are currently supported as transfer agents: ■ MediaGrid ContentBridge with remote Media API enabled. ■ Spectrum MediaDirector (firmware version 5.1 or later)

Return value

Value	Description
bool	Returns **true** if the specified host was contacted and is ready to perform the transfer. Returns **false** if the specified host could not be contacted or cannot perform the transfer for whatever reason.

Remarks

Must preceed any other calls related to a transfer.

Following the call to setRemoteHost the application will set up the transfer by calling the usual OmMediaCopier methods. In this way, if the setRemoteHost call failed or was omitted, the transfer will be carried as a local transfer, with data passing through the system where the application runs, as it is done in the current release of the API.

If setRemoteHost succeeds, there will be a restriction on the types of filenames that can be passed as source and destination for the transfer. When the transfer agent is a MediaGrid ContentBridge, the accepted formats are:

- \\aa.bb.cc.dd\some\path\clip.ext
- \\hostname\some\path\clip.ext
- //aa.bb.cc.dd/some/path/clip.ext
- //hostname/some/path/clip.ext
- /mnt/aa.bb.cc.dd/some/path/clip.ext

Where aa.bb.cc.dd and hostname are the IP address or hostname of the server where the clip is located.

When the transfer agent is a Spectrum MediaDirector the accepted formats are:

- • /some/path/clip.ext
- • mgfs:///some/path/clip.ext
- • mgfs://user:password@server/some/path/clip.ext

The first format should be used to reference files local to the Spectrum file system and the other two formats should be used for files residing in a remote MediaGrid server. The username, password, and server address configured through SystemManager will be used by default when they are omitted in the filename.

Filenames that do not follow any of the formats described above will not be accepted and will return an error.

To begin the transfer, the application will call OmMediaCopier::copy() as usual. The copy() call will block until the transfer is complete. Note that on a remote transfer blocking is not necessary, but it is done to remain compatible with local transfers.

The application can call OmMediaCopier::getProgress() on a different thread to monitor the transfer as it would do with a local transfer.

Limitations

None to note.

bool initializeRemoteHost ()

Specifies the initialization of a remote media server causing any ongoing transfers on that server to abort immediately.

Parameters

None.

Return value

Returns true if the remote host was initialized, false if the host could not be contacted or could not be initialized.

Remarks

This call may be useful in cases where the client application crashed or exited unexpectedly leaving one or more orphaned transfer tasks running on a remote server. The application can call this function after it is restarted to ensure that the host goes back to a clean state.

Limitations

A limitation of this function is that in a situation where there are several clients sending transfers to the same host all transfers will be aborted, even those that were started by a different client.

bool getDstInfo(OmMediaInfo&)

Description

Obtains the summary information for the destination movie. This changes as source tracks are added and parameters are changed.

Parameters

Parameter	Description
OmMediaInfo*&*	Specifies the structure into which movie information is to be placed.

Return value

Value	Description
bool	Returns **true** if valid data is returned. Returns **false** if setDestination has not been called or if it fails.

Remarks

Summary information changes as source tracks are added and parameters are changed.

Limitations

None to note.

See also

getMovieInfo, *OmMediaInfo*

bool get DstTrackInfo(uint trackNum, OmMediaSummary&)

Description

Obtains information about the specified destination track.

Parameters

Parameter	Description
uint trackNum	A numeric value that identifies the track in the destination file to be used. Tracks are zero based.
OmMediaSummary&	Specifies the structure into which track information is to be placed.

Return value

No values are returned.

Remarks

None to note.

Limitations

If the track does not exist, the results in the *OmMediaSummary* structure are unchanged.

bool get DstTrackInfo1(uint trackNum, OmMediaSummary1&)

Description

Obtains information about the specified destination track.

Parameters

Parameter	Description
uint trackNum	A numeric value that identifies the track in the destination file to be used. Tracks are zero based.
OmMediaSummary1&	Specifies the structure into which track information is to be placed.

Return value

No values are returned.

Remarks

None to note.

Limitations

If the track does not exist, the results in the *OmMediaSummary1* structure are unchanged.

bool set MaxRecordAge(uint seconds)

Description

Specifies the number of seconds a movie that is recording needs to be watched for changes, before assuming the recording has ended.

Parameters

Parameter	Description
uint seconds	A numeric value that specifies the number of seconds a movie that is recording needs to be watched for changes, before assuming the recording has ended.

Return value

No values are returned.

Remarks

The default value is 20 seconds.

Limitations

None.

bool copy()

Description

Performs the copy creating the new output movie and new media files as specified by the copy type.

Parameters

None.

Return value

Value	Description
bool	Returns **true** on success, or **false** if the destination movie does not support the specified essence files.

Remarks

An attempt is made to preserve the DefaultIn and DefaultOut values for a copied movie. It is assumed that if the DefaultIn value is zero, and the DefaultOut value is the end of the movie, then the values have not been set.

A movie is copied by taking a track or portion of a video or audio track from one or more source files.

The following rules apply to DefaultIn and DefaultOut values:

a. If a simple duplication of a movie is made, the DefaultIn and DefaultOut values are preserved.

b. If a movie is made by combining tracks from several source movies, and all movies have the same DefaultIn and DefaultOut values, or if only one of the source movies has a defaultIn and DefaultOut values, then the DefaultIn and DefaultOut will be set to that value.

Otherwise they will be set to the start and end of the resultant movie. In other words, DefaultIn and DefaultOut will be preserved, unless there are conflicting values to choose from.

c. If **Rule b** is satisfied, but the resultant movie does not contain all of the frames between DefaultIn and DefaultOut, DefaultIn and DefaultOut will be set to the start and end of the movie.

d. If the appendTrack() or appendTracks() function is used to combine more than one clip into any of the tracks, DefaultIn and DefaultOut will be set to the start and end of the resultant movie.

e. If the appendTrack() or appendTracks() function is used, but only a single clip is added to each of the tracks, **Rule b** applies.

f. If the resultant movie is a “clipped” version of the source movie, but still contains all of the frames between DefaultIn and DefaultOut, DefaultIn and DefaultOut will be adjusted so that they still point to the same frames.

Limitations

If disk space is insufficient, the copy fails. If files are locked the copy will fail.

OmCopyProgress getProgress()

void copyProgress(OmCopyProgress* progress)

Description

Gets the progress of the copy.

Parameters

Parameter	Description
OmCopyProgress* progress	Contains a pointer to the progress structure to be used.

Return value

Returns **OmCopyProgress**.

Remarks

This can be called from another thread while the copy is in progress. Note that it may not count linearly from one frame to the next because some tracks may be omitted if, for example, the media contains embedded audio.

Use **void copyProgress(OmCopyprogress* progress)** when working with a compiler other than Microsoft ®Visual Studio®.

Limitations

None to note.

void release()

Description

Frees up resources and closes open files.

Parameters

None.

Return value

No values returned.

Remarks

If you plan on holding on to the *OmMediaCopier* instance after you are done performing queries, then you should call *release()* when you are done to free up the resources and close open files.

Limitations

None to note.

void cancel()

Description

Cancels a copy.

Parameters

None.

Return value

No values returned.

Remarks

If a call to *copy()* is in progress, a call to *cancel()* by some other thread will cause the copy to end at the next frame. The movie at that point will be mostly coherent, although the length of the tracks may differ by one frame.

Limitations

None to note.

void setClipProperties (const char *clipProperties)

Description

Controls the format of the destination clip when several formats are available.

Parameters

Parameter	Description
const char *clipProperties	Specifies the desired properties for the destination clip.

Return value

No values returned.

Remarks

"The following clip formats are supported:

- "qt7": If the clip has MPEG media, it generates a wrapper that fully conforms to the QuickTime standard. This setting provides compatibility with Final Cut Pro for some media formats. Spectrum software prior to release 4.6 may not be able to play clips generated with this setting. For clips with video formats other than MPEG this setting is ignored.
- "qt6": If the clip has MPEG media, it generates a wrapper that is fully backwards compatible with Spectrum software prior to 4.6. For clips with video formats other than MPEG this setting is ignored.
- "d10": Generate an MXF OP1a eVTR wrapper. For clips that do not have the proper video and audio formats, this setting is ignored.
- "rdd9": Generate an MXF OP1a RDD9 compliant wrapper, compatible with Sony XDCAM-HD and XDCAM-HD422 products. For clips that do not have the proper video and audio formats, this setting is ignored.
- "opZeroTypeA": Generate an MXF Op0TypeA wrapper.

For QuickTime clips, if this function is not called, the API will choose the most appropriate wrapper format automatically using the following criteria:

- If at least one of the source clips has a QT7 wrapper, then the output will be wrapped with QT7.

- If none of the source clips has a QT7 wrapper, then the output will be wrapped with the Omneon backwards compatible wrapper by default. This default can be changed with the OmQtChooser application distributed with the Media API.

Limitations

None to note.

Query Movie Properties Class (OmMediaQuery)

Description

This class is used to query and change movie properties functions.

Remarks

After specifying a file, with **setFile**, an instance of this class provides access to the metadata, such as referenced essence files and size and location of individual samples. The default in and out point can be changed, and the movie can be renamed. This can result in the renaming of all the reference essence files as well.

Functions of the OmMediaQuery Class

The following are functions of the OmMediaQuery Class:

- *OmMediaQuery()*
- *~OmMediaQuery()*
- *bool setFile(const char * path, bool writable = false)*

bool setFile(const wchar_t path, bool writable = false)*

- *bool getMovieInfo(OmMediaInfo&)*
- *bool getTrackInfo(uint trackNum, OmMediaSummary&)*
- *bool getTrackInfo1(uint trackNum, OmMediaSummary1&)*
- *bool getSampleInfo(uint trackNum, int frameNum, OmMediaSample&)*
- *OmMediaId getMediaId(uint mediaIndex)*
- *bool getPath(OmMediaId, char * path, uint maxPathBytes)*

bool getPath(OmMediaId, wchar_t path, uint maxPathBytes)*

- *bool setDefaultIn(uint frameNum)*
- *bool setDefaultOut (uint frameNum)*
- *bool getFirstFrame(uint &frameNum) const*
- *bool setFirstFrame(uint frameNum)*
- *bool getStartTimeCode (uint &hour, uint &min, uint &sec, uint &frame) const*
- *bool setStartTimeCode (uint hour, uint min, uint sec, uint frame)*
- *bool getDropFrame (bool &dropFrame) const*
- *bool setDropFrame (bool dropFrame)*
- *bool getFrameData(uint trackNum, uint startFrame, uint numFrames,OmMediaClipDataType dataType, unsigned char *pData, uint *pDataSize)*
- *bool setFrameData(uint trackNum, uint startFrame, uint numFrames,OmMediaClipDataType dataType, unsigned char *pData, uint *pDataSize)*
- *const OmMediaPropertyList* getProperties() const*

const OmMediaPropertyList getProperties(uint trackNum) const*

const OmMediaPropertyList getAllProperties() const*

const OmMediaPropertyList getAllProperties(uint trackNum) const*

- *bool setProperty(const OmMediaProperty&)*
- *bool deleteProperty(const OmMediaProperty&)*
- *bool rename(const char * newPath, bool renameMedia = false)*

bool rename(const wchar_t newPath, bool renameMedia = false)*

- *bool remove()*
- *void flush()*
- *const char*getUmid() const*
- *bool generateUmid()*
- *void release()*
- *unsigned char getVideoAfd(uint trackNum = ~0) const*
- *bool setVideoAfd(unsigned char afd, uint trackNum = ~0)*
- *bool OmMediaQuery::isOmneonMovie()*
- *OmSwVersion OmMediaQuery::getOmneonCreateVersion()*
- *OmSwVersion OmMediaQuery::getThisSwVersion()*
- *OmSwVersion getOmneonCreateVersion() const*
- *bool isOmneonMovie() const*

OmMediaQuery()

Description

Constructs a new *OmMediaQuery*.

Parameters

None.

Return value

No values returned.

Remarks

None to note.

Limitations

None to note.

~OmMediaQuery()

Description

Destructor for *OmMediaQuery*.

Parameters

None.

Return value

No values returned.

Remarks

None to note.

Limitations

None to note.

bool setFile(const char * path, bool writable = false)

bool setFile(const wchar_t* path, bool writable = false)

Description

Opens the specified media or movie path, and if successful, makes that file the one used in other methods.

Parameters

Parameter	Description
const char * path **const wchar_t* path**	A text string containing a pointer to the name of the file to open.
bool writable = false	**True** allows OmMediaQuery to change the contents of the file. For example: *SetProperty()*, *SetDefaultIn()*, or *SetDefaultOut()*.

Return value

Value	Description
bool	Returns **true** on success; **false** if the file is corrupt, missing, or unrecognized, or if writable was **true** but the file could not be opened with write permission.

Remarks

Any previously held movie files are closed and released. Writable must be specified as **true** to use any of the set methods, such as in **setDefaultIn** and **setDefaultOut**.

Limitations

None to note.

See also

setProperty, *setDefaultIn*, **setDefaultOu***t*

bool getMovieInfo(OmMediaInfo&t)

Description

Gets movie information associated with the current file.

Parameters

Parameter	Description
OmMediaInfo&t	Identifies the structure into which the movie information is to be placed.

Return value

Value	Description
bool	Always returns **true** unless **setFile** was unsuccessful.

Remarks

On success, the *OmMediaInfo* instance is filled in.

Limitations

None to note.

See also

setFile, *OmMediaInfo*

bool getTrackInfo(uint trackNum, OmMediaSummary&)

Description

Gets summary information for the specified track.

Parameters

Parameter	Description
uint trackNum	A numeric value that specifies the track number. Tracks are zero based.
OmMediaSummary&	Identifies the structure into which the track information is to be placed.

Return value

Value	Description
bool	Returns **true** on success. Returns **false** if the track number is greater than *OmMediaInfo::numTracks*, or if the track does not exist.

Remarks

None to note.

Limitations

None to note.

See also

OmMediaInfo, **OmMediaSummary**

bool getTrackInfo1(uint trackNum, OmMediaSummary1&)

Description

Gets summary information for the specified track.

Parameters

Parameter	Description
uint trackNum	A numeric value that specifies the track number. Tracks are zero based.
OmMediaSummary1&t	Identifies the structure into which the track information is to be placed.

Return value

Value	Description
bool	Returns **true** on success. Returns **false** if the track number is greater than *OmMediaInfo::numTracks*, or if the track does not exist.

Remarks

None to note.

Limitations

None to note.

See also

OmMediaInfo, **OmMediaSummary**

bool getSampleInfo(uint trackNum, int frameNum, OmMediaSample&t)

Description

Gets detailed information about the specified frame.

Parameters

Parameter	Description
uint trackNum	A numeric value that specifies the track number. Tracks are zero based.
int frameNum	A numeric value that specifies the number assigned to the frame.
OmMediaSample&t	Identifies the structure into which track information is to be placed.

Return value

Value	Description
bool	Returns **true** on success. Returns **false** if the track number is greater than OmMediaInfo::numTracks or if the frame number is out of bounds.

Remarks

A frame of audio consists of all samples associated with the corresponding video frame.

The frame number may be negative for video tracks only. Thus the bounds for valid frame numbers F: in <= F < out, where in is OmMediaInfo::firstFrame - numPrecharge and out is OmMediaInfo::lastFrame.

If getSampleInfo() is called on a file which is an audio essence file (e.g. a WAVE or AIFF file), getSampleInfo() will assume that the frame rate is 29.97 (NTSC) for purposes of calculating the sample location and size.

Limitations

None to note.

See also

OmMediaSample

OmMediaId getMediaId(uint mediaIndex)

Description

Gets the media id with the specified index.

Parameters

Parameter	Description
uint mediaIndex	A numeric value that specifies the media ID.

Return value

Value	Description
OmMediaId	Returns **non-zero** on success, or returns **0** if mediaIndex > OmMediaInfo::numMedia.

Remarks

An *OmMediaId* is a pointer to an opaque structure, and a unique id exists for each unique reference file in the movie. Note that media ids are only unique within a single movie, not across all movies.

Limitations

None to note.

bool getPath(OmMediaId, char * path, uint maxPathBytes)

bool getPath(OmMediaId, wchar_t* path, uint maxPathBytes)

Description

Gets the pathname associated with the specified *OmMediaId*.

Parameters

Parameter	Description
OmMediaId	Specifies the Media ID for which a path is desired.
wchar * path **wchar_t* path**	A text string to receive the pathname of the file.
uint maxPathBytes	A numeric value that specifies the maximum number of bytes for the pathname associated with the specified *OmMediaId*. The upper limit is 260 bytes.

Return value

Value	Description
bool	Returns **true** on success. Returns **false** if the path is longer than *maxPathBytes* or the media id is not valid for the current movie.

Remarks

None to note.

Limitations

None to note.

See also

getMediaId

bool setDefaultIn(uint frameNum)

Description

Sets the default in frame for the movie.

Parameters

Parameter	Description
uint frameNum	A numeric value that specifies the frame number for the default in frame.

Return value

Value	Description
bool	Returns **true** on success. Returns **false** if frameNum < *OmMediaInfo::firstFrame*, or after the default out, or the movie does not support default in/out. For example, QuickTime movies can store default in and out (known as select in and out), but DV files cannot.

Remarks

Note that the movie will not be updated until (a) new movie is specified with **setFile**, or (b) **flush** is called, or (c) the **OmMediaQuery** instance is destroyed.

Limitations

None to note.

See also

setDefaultOut

bool setDefaultOut (uint frameNum)

Description

Sets the default out frame for the movie.

Parameters

Parameter	Description
uint frameNum	A numeric value that specifies the frame number for the default out frame.

Return value

Value	Description
bool	Returns **true** on success. Returns **false** if frameNum > OmMediaInfo::lastFrame, or before the default in, or the movie does not support the notion of default in/out.

Remarks

None to note.

Limitations

None to note.

See also

setDefaultIn

bool getFirstFrame(uint &frameNum) const

Description

Gets the property list for the current movie.

Parameters

Parameter	Description
uint &frameNum	A numeric value that specifies the first frame number.

Return value

Value	Description
bool	Returns **true** on success.

Remarks

None to note.

Limitations

None to note.

bool setFirstFrame(uint frameNum)

Description

Sets the starting frame number of the movie.

Parameters

Parameter	Description
uint frameNum	A numeric value that specifies the first frame number.

Return value

Value	Description
bool	Returns true on success. Returns false if frameNum > OmMediaInfo::lastFrame, or before the default in, or the movie does not support the notion of default in/out.

Remarks

Setting the starting frame number to a value greater than zero causes the movie to be loaded onto the timeline at a time later than time = zero.

Limitations

None to note.

bool getStartTimeCode (uint &hour, uint &min, uint &sec, uint &frame) const

Description

Gets the timecode of the start of the movie.

Parameters

Parameter	Description
uint &hour	A numeric value that specifies the hour at which the movie starts.
uint &min	A numeric value that specifies the minutes after the hour at which the movie starts.
uint &sec	A numeric value that specifies the seconds after the hour at which the movie starts.
uint &frame	A numeric value that specifies the frame within the second.

Return value

Value	Description
bool	Returns **true** on success.

Remarks

None to note.

Limitations

None to note.

bool setStartTimeCode (uint hour, uint min, uint sec, uint frame)

Description

Sets the timecode of the start of the movie.

Parameters

Parameter	Description
uint hour	A numeric value that specifies the hour at which the movie starts.
uint min	A numeric value that specifies the minutes after the hour at which the movie starts.
uint sec	A numeric value that specifies the seconds after the hour at which the movie starts.
uint frame	A numeric value that specifies the frame within the second.

Return value

Value	Description
bool	Returns true on success. Returns false if invalid timecode.

Remarks

This modifies the timecode track of the movie to start at a time other than the default of 00:00:00:00.

Limitations

None to note.

bool getDropFrame (bool &dropFrame) const

Description

Gets the value of the dropframe flag from the start timecode.

Parameters

Parameter	Description
bool &dropFrame	Provides the value of the drop frame flag.

Return value

Returns **true** if successful, **false** otherwise.

Remarks

Refer to *SMPTE 12M-1999 Television, Audio and Film – Time and Control Code* available at: *http://www.smpte.org/home* for additional information.

Limitations

None to note.

bool setDropFrame (bool dropFrame)

Description

Sets the value of the dropframe flag of the start timecode.

Parameters

Parameter	Description
bool dropFrame	A value that specifies that a dropframe is inserted.

Return value

Returns **true** if successful, **false** otherwise.

Remarks

Changing this flag may impact the way timecodes obtained from the movie are interpreted on subsequent playback, depending on the timecode generator counter mode used for playback. Setting the dropframe flag to true does not cause frames to be dropped. Refer to *SMPTE 12M-1999 Television, Audio and Film – Time and Control Code* available at: *http://www.smpte.org/home* for additional information.

Limitations

Changing the dropframe mode of the start timecode does not change the dropframe mode of the per-frame timecodes. Doing so may produce an inconsistent movie file. The setFrameData() API provides a way to change per-frame timecodes.

bool getFrameData(uint trackNum, uint startFrame, uint numFrames,OmMediaClipDataType dataType, unsigned char *pData, uint *pDataSize)

Description

Extracts data from a clip, such as timecode, closed caption, or VANC.

Parameters

Parameter	Description
uint trackNum	A numeric value that specifies the track number. Tracks are zero based.
uint startFrame	A numeric value that provides the frame number to start data extraction from the clip. Frames are referenced in display order.
uint numFrames	A numeric value that determines the number of frames to extract from the clip. This value can range from 1 to 512.
OmMediaClipDataType dataType	Determines the type of data to extract, as specified by enum OmMediaClipDataType.
unsigned char *pData	A pointer to the buffer where the extracted data is returned. This buffer must be allocated by the caller and must be large enough to hold all of the returned data. If extracting VANC, the data in the buffer will be formatted as Omneon Neutral VANC.
uint *pDataSize	A pointer to the size of the pData buffer. It must be set by the caller to point to the size (in bytes) of the pData buffer. Upon return, the value will be updated to the amount of data returned.

Return value

Value	Description
bool	Returns true on success or false on failure.

Remarks

This function can extract a number of frames worth of CC data starting at a specified frame from a specified clip. Extracting all of the CC data from any clip more than a few seconds long requires multiple calls to getFrameData. Multiple calls can be made to extract CC data from random clips at random frame start locations, however, the best performance will be achieved by extracting frame sequential chunks of CC data from a single clip. For example: to extract all CC data from a 10,000 frame long clip, make 50 calls to getFrameData, each call will extract 200 frames worth of CC starting at frame locations 0, 200, 400, 600... 9800.

This function works on frames in display order and handles any frame reordering that is required for MPEG. For example, an API call for frame 0 will be for the first display frame, rather than the first frame in the file.

Limitations

None to note.

bool setFrameData(uint trackNum, uint startFrame, uint numFrames,OmMediaClipDataType dataType, unsigned char *pData, uint *pDataSize)

Description

Inserts timecode or closed caption data into a clip or VANC.

Parameters

Parameter	Description
uint trackNum	A numeric value that specifies the track number. Tracks are zero based.
uint startFrame	A numeric value that provides the frame number to start data insertion into the clip. Frames are referenced in display order.
uint numFrames	A numeric value that determines the number of frames to insert into the clip. This value can range from 1 to 512.
OmMediaClipDataType dataType	Determines the type of data to insert, as specified by enum OmMediaClipDataType.
unsigned char *pData	A pointer to the buffer where the data is to be inserted is stored. When inserting VANC, the data in the buffer must be formatted as Omneon Neutral VANC.
uint *pDataSize	A pointer to the size of the pData buffer. It must be set by the caller to point to the size (in bytes) of the pData buffer. Upon return, the value will be updated to the amount of data inserted.

Return value

Value	Description
bool	Returns true on success or false on failure.

Remarks

This function can insert a number of frames worth of CC data starting at a specified frame for a specified clip. Inserting all of the CC data into any clip more than a few seconds long requires multiple calls to setFrameData. Multiple calls can be made to insert CC data into random clips at random frame start locations, however, the best performance will be achieved by inserting frame sequential chunks of CC data into a single clip. For example: to insert all CC data from a 10,000 frame long clip, make 50 calls to setFrameData each call inserting 200 frames worth of CC starting at frame locations 0, 200, 400, 600... 9800.

This function works on frames in display order and handles any frame reordering that is required for MPEG. For example, an API call for frame 0 will be for the first display frame, rather than the first frame in the file.

Limitations

None to note.

const OmMediaPropertyList* getProperties() const

const OmMediaPropertyList* getProperties(uint trackNum) const

const OmMediaPropertyList* getAllProperties() const

const OmMediaPropertyList* getAllProperties(uint trackNum) const

Description

Gets the property list for the specified track number.

Parameters

Parameter	Description
uint trackNum	A numeric value that specifies the number assigned to the specified track. Tracks are zero based.

Return value

Value	Description
OmMediaPropertyList*	Always returns a non-null pointer, although it may be empty if the track does not contain any string properties, or no movie is currently defined.

Remarks

If you change the current movie, the property list may also change. getProperties() returns a property list with string values; getAllProperties() returns a property list with all the values.

Limitations

None to note.

bool setProperty(const OmMediaProperty&t)

bool setProperty(uint trackNum, const OmMediaProperty&t)

Description

Sets the specified property in the current movie.

Parameters

Parameter	Description
const OmMediaProperty&t	Specifies the structure into which the property will be stored.
uint trackNum	A numeric value that specifies the number assigned to the specified track. Tracks are zero based.

Return value

Value	Description
bool	Returns **true** on success. Returns **false** if the movie could not be updated, or it is a read-only, or there is a conflict with some other internal property of the same name but with a different type.

Remarks

If the property already exists, it will be replaced with a new value.

Limitations

None to note.

bool deleteProperty(const OmMediaProperty&t)

bool deleteProperty(uint trackNum, const OmMediaProperty&t)

Description

Deletes a property on the currently defined movie.

Parameters

Parameter	Description
uint trackNum	A numeric value that specifies the number assigned to the specified track. Tracks are zero based.
const OmMediaProperty&t	Specifies the structure where the property is located

Return value

Returns **false** if no movie is defined, or it is read-only, or there is a conflict with another internal property of the same name but with a different type.

Remarks

None to note.

Limitations

None to note.

bool rename(const char * newPath, bool renameMedia = false)

bool rename(const wchar_t* newPath, bool renameMedia = false)

Description

Changes the name of the movie to newPath; if renameMedia is true, any media files referenced by the movie will be renamed to match the movie.

Parameters

Parameter	Description
const char * newPath **const wchar_t* newPath**	A text string containing a pointer to the name of the file to open.
bool renameMedia = false	If renameMedia is **false**, no referenced media will be renamed. Thus, if renameMedia is **true**, all referenced media will be renamed.

Return value

Value	Description
bool	Returns **true** on success. Returns **false** if the movie could not be updated or the new name already exists or is invalid.

Remarks

Relative pathnames are valid.

Limitations

None to note.

bool remove()

Description

Removes the movie along with any media files referenced by the movie.

Parameters

None.

Return value

Value	Description
bool	Returns **true** on success. Returns **false** if current movie is not valid or you do not have permission to remove the files.

Remarks

None to note.

Limitations

None to note.

void flush()

Description

Flushes changes to the movie made by the set operations out to disk.

Parameters

None.

Return value

No values returned.

Remarks

None to note.

Limitations

None to note.

const char*getUmid() const

Description

Gets the UMID for the current movie.

Parameters

None.

Return value

Value	Description
char*	Returns the movie's UMID (if present), or a null pointer if no UMID is present or no movie is currently defined.

Remarks

The UMID matches the format specified in SMPTE 330m, which is a hexadecimal string prefixed with "0x".

Limitations

None to note

bool generateUmid()

Description

Generates a new UMID for the current movie.

Parameters

None.

Return value

Value	Description
bool	Returns **true** on success. Returns **false** if current movie is not valid, or is read-only.

Remarks

None to note.

Limitations

None to note.

void release()

Description

Frees up resources and closes open files.

Parameters

None.

Return value

No values returned.

Remarks

If you plan on holding on to the *OmMediaQuery* instance after you are done performing queries, then you should call *release()* when you are done to free up the resources and close open files.

Limitations

None to note.

unsigned char getVideoAfd(uint trackNum = ~0) const

Description

Interface to get the active format description (AFD) of the clip, which is represented here as a single byte of unsigned char type.

Parameters

Parameter	Description
trackNum	Indicates what track to use. If the default of ~0 is given then the first video track in the clip will be used.

Return value

Value	Description
unsigned char	The AFD of the clip. See SMPTE 2016-1-2009 Table 4 for the format of the AFD byte.

Limitations

None to note.

bool setVideoAfd(unsigned char afd, uint trackNum = ~0)

Description

Interface to set the active format description (AFD) of the clip.

Parameters

Parameter	Description
afd	The AFD to write to the clip. See SMPTE 2016-1-2009 Table 4 for the format of the AFD byte.
trackNum	Indicates what track to use. If the default of ~0 is given then the first video track in the clip will be used.

Return value

Value	Description
bool	Returns **true** on success. Returns **false** if the track number is invalid, if the clip does not have a video track, or if, for any reason, the AFD byte could not be written to the clip.

Limitations

None to note.

bool OmMediaQuery::isOmneonMovie()

Description

Function to determine if a clip was created by Omneon.

Parameters

None.

Return value

Value	Description
bool	Returns **true** if the clip was created by Omneon. Returns **false** if the clip was not created by Omneon.

Remarks

None to note

Limitations

None to note.

OmSwVersion OmMediaQuery::getOmneonCreateVersion()

Description

Determines the version of the Omneon Media Layer used to create a clip.

Parameters

None.

Return value

Returns the Omneon Media Layer version used to create a clip. If the version could not be determined or if the clip was not created by Omneon, this function returns an OmSwVersion with a value of 255.255.255.255.

Remarks

None to note.

Limitations

If the clip was created with a Spectrum version prior to 5.3, this function will be unable to determine the software version used to create the clip and will return an OmSwVersion with a value of 255.255.255.255.

OmSwVersion OmMediaQuery::getThisSwVersion()

Description

Determines the version of Omneon Media Layer currently in use.

Parameters

None.

Return value

OmSwVersion - A class that represents an Omneon software version.

Remarks

None to note.

Limitations

None to note.

See also

OmSwVersion getOmneonCreateVersion() const, *bool isOmneonMovie() const*.

OmSwVersion getOmneonCreateVersion() const

Description

Returns the Omneon Media Layer version user to write a movie, if known. If the version could not be determined, or if the file was not written by Omneon, then OmSwVersion::isValid() returns false.

Parameters

None.

Return value

OmSwVersion - A class that represents an Omneon software version.

Remarks

None to note.

Limitations

None to note.

See also

OmSwVersion OmMediaQuery::getThisSwVersion(), *bool isOmneonMovie() const*.

bool isOmneonMovie() const

Description

Returns true if this movie was written by Omneon.

Parameters

None.

Return value

bool - Returns true if the movie was written by Omneon, false otherwise.

Remarks

If this function returns true then you can use getOmneonCreateVersion to find out what version of Omneon software wrote the movie.

Limitations

None to note.

See also

OmSwVersion OmMediaQuery::getThisSwVersion(), *OmSwVersion getOmneonCreateVersion() const*.

Media Property Class (OmMediaProperty)

Description

This class provides data for the *Media Property List Class (OmMediaPropertyList)*.

Remarks

This class is *only* used by the *Media Property List Class (OmMediaPropertyList)*.

Functions of the Media Property Class

The following are functions of the OmMediaProperty Class:

- *OmMediaProperty (const char* name, const char* value)*
- *OmMediaProperty (const char* name, const void* value, uint valueLen, OmMediaPropertyType type)*
- *~OmMediaProperty()*
- *const char * getValue() const*
- *const char* getName() const*
- *const uint getValueLength()*
- *const OmMediaPropertyType getType()*
- *void set(const char * name, const char * value)*

OmMediaProperty (const char* name, const char* value)

Description

Constructs a new *OmMediaProperty* with the specified name and value.

Parameters

Parameter	Description
const char* name	Shows the name (keyword) of the property.
const char* value	Shows the value associated with the name.

Return value

No values returned.

Remarks

None to note.

Limitations

None to note.

OmMediaProperty (const char* name, const void* value, uint valueLen, OmMediaPropertyType type)

Description

Constructs a new *OmMediaProperty* with the specified name, value, and property type.

Parameters

Parameter	Description
const char* name uint valueLen	Contains a pointer to the name of the property which is to be copied.
const void* value	Contains a pointer to the value of the property.
uint valueLen	Contains a value indicating the property length.
OmMediaPropertyType type	Contains the property type.

Return value

Returns the length of the media value.

Remarks

None to note.

Limitations

None to note.

~OmMediaProperty()

Description

Destructor for *OmMediaProperty*.

Parameters

None.

Return value

No values returned.

Remarks

Note that properties returned by *OmMediaPropertyList::get()* are constant and cannot be destroyed.

Limitations

None to note.

const char * getValue() const

Description

Gets the string value of the property.

Parameters

None.

Return value

Returns the value of the property.

Remarks

None to note.

Limitations

None to note.

See also

getName

const char* getName() const

Description

Gets the string name of the property.

Parameters

None.

Return value

Returns the name of the property.

Remarks

None to note.

Limitations

None to note.

See also

getValue

const uint getValueLength()

Description

Gets the length of the property's value.

Parameters

None.

Return value

Returns the length of the property's value.

Remarks

None to note.

Limitations

None to note.

See also

getValue

const OmMediaPropertyType getType()

Description

Gets the type of the property.

Parameters

None.

Return value

Returns the value type of the property.

Remarks

None to note.

Limitations

None to note.

See also

getValue

void set(const char * name, const char * value)

Description

Sets the name and value of an existing *OmMediaProperty* instance.

Parameters

Parameter	Description
const char * name	Shows the new name of the *OmMediaProperty*.
const char * value)	Shows the new value of the *OmMediaProperty*.

Return value

None returned.

Remarks

Properties returned by *OmMediaPropertyList::get()* are constant and cannot be set. To set a property in an omMedia use *omMediaPropertyList::set()*.

Limitations

None to note.

Media Property List Class (OmMediaPropertyList)

Description

This class is used to create a list of OmMediaProperty.

Remarks

Given an instance of *OmMediaQuery()*, an instance of an OmMediaPropertyList can be obtained with one of the two OmMediaQuery::getProperties() methods. This list and its elements can be queried as long as the originating OmMediaQuery instance still exists. Note that you can set a property in a OmMediaPropertyList; however, it does not affect the movie. This method is used internally to assemble copies of the user data values. To change properties in the movie, use OmMediaQuery::setProperty().

Functions of the Media Property List Class

The following are functions of the *OmMediaPropertyList* class:

- *OmMediaPropertyList()*
- *~OmMediaPropertyList()*
- *uint getPropertyCount() const*

- *const OmMediaProperty* get(uint index) const*
- *const OmMediaProperty* get(const char *) const*
- *bool set(const OmMediaProperty&)*

OmMediaPropertyList()

Description

Constructs a new instance of *OmMediaPropertyList*.

Parameters

None.

Return value

No values returned.

Remarks

Generally, it is unnecessary to create a new *OmMediaPropertyList* because one is provided by the call to *OmMediaQuery::getProperties()*.

Limitations

None to note.

~OmMediaPropertyList()

Description

Destructor for *OmMediaPropertyList*.

Parameters

None.

Return value

No values returned.

Remarks

None to note.

Limitations

None to note.

uint getPropertyCount() const

Description

Gets the number of properties specified in the list.

Parameters

None.

Return value

Returns the number of properties in the list.

Remarks

None to note.

Limitations

None to note.

const OmMediaProperty* get(uint index) const

Description

Gets the property at the specified index.

Parameters

Parameter	Description
uint index	A numeric value that specifies which *OmMediaProperty* to return.

Return value

Returns a pointer to the desired *OmMediaProperty* or null if the index is out of range.

Remarks

None to note.

Limitations

None to note.

const OmMediaProperty* get(const char *) const

Description

Gets the property with the specified name.

Parameters

Parameter	Description
const char*	Specifies the name of the desired *OmMediaProperty*.

Return value

Returns a pointer to the *OmMediaProperty* or returns **null** if no property exists that matches the name.

Remarks

None to note.

Limitations

None to note.

bool set(const OmMediaProperty&)

Description

Sets the specified property in the list.

Parameters

Parameter	Description
const OmMediaProperty&t	Specifies the new or existing name and value of the *OmMediaProperty*.

Return value

Value	Description
bool	Returns **true** on success. Returns **false** if the name or value is null.

Remarks

If a property already exists with the given name, its name will be replaced with the new value. If no property matches the name, the new property will be added to the list.

Limitations

None to note.

Software Version Class (struct OmSwVersion)

Description

This class provides a representation of an Omneon software version.

Remarks

None to note.

Functions of the Omneon Software Version Class

The following are functions of the *struct OmSwVersion* class:

- *uint getMajor() const*
- *uint getMinor() const*
- *uint getService() const*
- *uint getHotfix() const*
- *bool isValid() const*
- *bool isNotValid() const*

uint getMajor() const

Description

Returns the version number of a major Spectrum software release, indicated by the first digit in a version number.

Parameters

None

Return value

Value	Description
m_major	Specifies the version number of a major software release.

Remarks

None to note.

Limitations

None to note.

uint getMinor() const

Description

Returns the version number of a minor Spectrum software release, indicated by the second digit in a version number.

Parameters

None

Return value

Value	Description
m_minor	Specifies the version number of a minor software release.

Remarks

None to note.

Limitations

None to note.

uint getService() const

Description

Returns the version number of a Spectrum service software release, indicated by the third digit in a version number.

Parameters

None

Return value

Value	Description
m_service	Specifies the version number of a service software release.

Remarks

None to note.

Limitations

None to note.

uint getHotfix() const

Description

Returns the version number of a Spectrum hotfix software release, indicated by the fourth digit in a version number.

Parameters

None

Return value

Value	Description
m_hotfix	Specifies the version number of a hotfix software release.

Remarks

None to note.

Limitations

None to note.

bool isValid() const

Description

Determines if the version is a valid Omneon software version number.

Parameters

None

Return value

Value	Description
bool	Returns **True** if a version number is valid. Returns **False** if the software version is invalid or could not be determined.

Remarks

None to note.

Limitations

None to note.

bool isNotValid() const

Description

Determines if the version is an invalid Omneon software version number.

Parameters

None

Return value

Value	Description
bool	Returns **True** if a version number is invalid or could not be determined. Returns **False** if the software version is valid.

Remarks

None to note.

Limitations

None to note.

Data Types

Structures and Enums are used throughout the interface, as input and/or output arguments.

Description of Structures

Structures are used to contain groups of related information. The following Structures are used:

- *Struct OmCopyProgress*
- *Struct OmMediaInfo*
- *Struct OmMediaSample*
- *Struct OmMediaSummary*
- *Struct OmMediaSummary1*
- *Struct MediaSummaryName*
- *Struct MPEG*
- *Struct Audio*
- *Struct VBI*
- *Struct OmTcData*

Description of Enums

Enums are used to restrict an argument to a known set of values. The following Enums are used:

- *Enum OmMediaClipDataType*
- *Enum OmMediaCopyType*
- *Enum OmMediaFileType*
- *Enum OmVideoFormat*
- *Enum OmVideoScan*

- *Enum OmFrameStruct*
- *Enum OmFieldRate*
- *Enum OmStreamType*
- *Enum OmFrameRate*
- *Enum OmFieldID*
- *Enum OmMediaType*
- *Enum {OmMediaNumTypes = 13}*
- *Enum OmVideoSampleRatio*
- *Enum OmVideoAspectRatio*
- *Enum OmAudioFormat*
- *Enum OmMediaPropertyType*

Struct OmCopyProgress

Description

Describes the number of frames already copied and the total number of frames to be copied.

```
struct OmCopyProgress {
uint nTotalFrames;
uint nComplete;
};
```

Descriptions of the *OmCopyProgress* structure fields are provided in the following table.

OmCopyProgress Structure Fields	Description
`uint nTotalFrames`	Equal to (out-in) *nTracks
`uint nComplete`	Frames completed out of total copy

Remarks

None to note.

Struct OmMediaInfo

Description

Provides general information about the file being created.

```
typedef struct OmMedaIdRec* OmMediaId;
struct OmMediaInfo {
uint firstFrame;
uint lastFrame;
uint defaultIn;
uint defaultOut;
uint numTracks;
uint numMedia;
uint numPrecharge;
OmFrameRate frameRate;
const char* clipProperties;
};
```

Descriptions of the *OmMediaInfo* structure fields are provided in the following table.

OmMediaInfo Structure Fields	Description
`uint firstFrame`	First recorded frame (may be non-zero)
`uint lastFrame`	Last recorded frame
`uint defaultIn`	Default in (inclusive, 1st frame to play)
`uint defaultOut`	Default out (excluding., 1st frame *not* played)
`uint numTracks`	# of video and audio tracks
`uint numMedia`	# of media files referenced by movie
`uint numPrecharge`	# of precharge frames before the first frame
`OmFrameRate frameRate`	Clip frame rate
`const char* clipProperties`	Describes features of the clip. “qt7” indicates an MPEG clip wrapped with the QT7 wrapper standard. “qt6” indicates an MPEG clip wrapped with and Omneon QuickTime wrapper. “d10” indicates a clip is in MXF OP1a eVTR format. Any other types of clips will have this field set to an empty string.

Remarks

None to note.

Struct OmMediaSample

Description

Describes a frame.

```
struct OmMediaSample {
int frameNum;
uint nBytes;
uint_64 offset;
OmMediaId id;
};
```

Descriptions of the *OmMediaSample* structure fields are provided in the following table.

OmMediaSample Structure Fields	Description
`int frameNum`	Frame number of the sample; may be negative
`uint nBytes`	Number of bytes in the sample
`uint_64 offset`	File offset for the sample
`OmMediaId id`	Media ID for the file containing the sample

Remarks

None to note.

Struct OmMediaSummary

Description

Provides summary information about a video or audio track.

```
struct OmMediaSummary {
OmMediaType type;
OmVideoSampleRatio vsr;
OmVideoAspectRatio aspect;
uint bitrate;
uint bitsPerUnit;
uint sampleRate;
uint channels;

union Specific {
struct Mpeg {
uint gopLength;
uint subGopLength;
} mpeg;
struct Audio {
OmAudioFormat format;
unsigned char bigEndian;
unsigned char sampleStride;
} audio;
struct Vbi {
uint lineMask;
} vbi;
} specific;
```

Descriptions of the *OmMediaSummary* structure fields are provided in the following table.

OmMediaSummary Structure Flelds	Description
`OmMediaType type`	As defined in enums
`OmVideoSampleRatio vsr`	As defined in enums
`OmVideoAspectRatio aspect`	As defined in enums
`uint bitrate`	Bitrate for the track; 0 if it is embedded audio
`uint bitsPerUnit`	16 or 24 for audio (unit is a sample); 8 or 10 for 601 video (unit is a pixel); otherwise 0
`uint sampleRate`	Media sample rate * 100; e.g. ntsc video is 2997. 48khz audio is 48000
`uint channels`	1 for video; N for audio union

Remarks

None to note.

Struct OmMediaSummary1

Description

Provides summary information about a video or audio track as in *Struct OmMediaSummary* and in addition includes width and height data.

```
struct OmMediaSummary1 {
OmMediaType type;
OmVideoSampleRatio vsr;
OmVideoAspectRatio aspect;
uint bitrate;
uint bitsPerUnit;
uint sampleRate;
uint channels;

union Specific {
struct Mpeg {
uint gopLength;
uint subGopLength;
unsigned short width;
unsigned short height;
} video;
struct Audio {
OmAudioFormat format;
unsigned char bigEndian;
unsigned char sampleStride;
} audio;
struct Vbi {
uint lineMask;
} vbi;
} specific;
```

Descriptions of the *OmMediaSummary1* structure fields are provided in the following table.

OmMediaSummary1 Structure Fields	Description
`OmMediaType type`	As defined in enums
`OmVideoSampleRatio vsr`	As defined in enums
`OmVideoAspectRatio aspect`	As defined in enums
`uint bitrate`	Bitrate for the track; 0 if it is embedded audio
`uint bitsPerUnit`	16 or 24 for audio (unit is a sample); 8 or 10 for 601 video (unit is a pixel); otherwise 0
`uint sampleRate`	Media sample rate * 100; e.g. ntsc video is 2997. 48khz audio is 48000
`uint channels`	1 for video; N for audio union

Remarks

None to note.

Struct MediaSummaryName

Description

Provides a mechanism which translates between enums and string representations.

```
struct MediaSummaryName {
static const char* get(OmMediaType);
static const char* get(OmStreamType);
static const char* get(OmVideoSampleRatio);
static const char* get(OmVideoAspectRatio);
static const char* get(OmFrameRate);

static OmMediaType getMediaType(const char*);
static OmVideoSampleRatio getVsr(const char*);
static OmVideoAspectRatio getVar(const char*);
static OmFrameRate getFrameRate(const char*)
};
```

Descriptions of the *Struct MediaSummaryName* structure fields are provided in the following table.

Struct MediaSummaryName Structure Fields	Description
`static const char* get(OmMediaType)`	Returns a string representation of an OnMediaType.
`static const char* get(OmStreamType)`	Returns a string representation of an OnStreamType.
`static const char* get(OmVideoSampleRatio)`	Returns a string representation of an OnVideoSampleRatio.
`static const char* get(OmVideoAspectRatio)`	Returns a string representation of an OnVideoAspectRatio.
`static const char* get(OmFrameRate)`	Returns a string representation of an OnFrameRate.
`static OmMediaType getMediaType(const char*)`	Returns an OmMediaType that matches the given string, but returns “OmMediaUnknown” if the string is not recognized.

Struct MediaSummaryName Structure Fields	Description
`static OmVideoSampleRatio getVsr(const char*)`	Returns an OmVideoSampleRatio that matches the given string, but returns “OmVideoSampleNone” if the string is not recognized.
`static OmVideoAspectRatio getVar(const char*)`	Returns an OmVideoAspectRatio that matches the given string, but returns “OmVideoAspectNone” if the string is not recognized.
`static OmFrameRate getFrameRate(const char*)`	Returns an OmFrameRate that matches the given string, but returns “OmFrameRateInvalid” if the string is not recognized.

Remarks

None to note.

Struct MPEG

Description

Provides video information if *OmMediaType* type is MPEG.

```
struct Mpeg {
uint gopLength;
uint subGopLength;
} mpeg;
```

Descriptions of the *MPEG* structure fields are provided in the following table.

MPEG Structure Fields	Description
`uint gopLength`	Length of the gop structure; usually 15 for MPEG, but is one for other formats
`uint subGopLength`	Length of the sub-gop; usually 3 for IBBPBBP... MPEG video; one for other formats

Remarks

None to note.

Struct Audio

Description

Provides audio information if *OmMediaType* type is audio.

```
struct Audio {
OmAudioFormat format;
unsigned char bigEndian;
unsigned char sampleStride;
} audio;
```

Descriptions of the *Audio* structure fields are provided in the following table.

Audio Structure Fields	Description
`OmAudioFormat format`	PCM, Raw, or AC3
`unsigned char bigEndian`	1 if data is Big Endian, 0 if data is little Endian
`unsigned char sampleStride`	Number of bytes to next sample

Remarks

None to note.

Struct VBI

Description

Provides VBI information if *OmMediaType* type is VBI.

```
struct Vbi {
uint lineMask;
} vbi;
```

Descriptions of the *VBI* structure field are provided in the following table.

VBI Structure Field	Description
`uint lineMask`	Bit mask indicating lines that are encoded

Remarks

None to note.

Struct OmTcData

Description

The OmTcData structure presents the raw 64 bits of the house time reading; of the 64 bits, 32 are user data and some of the rest are dedicated flag bits such as the Drop Frame flag. The OmTcData structure will be filled with 0xFF values if there is no valid VITC time available. The OmTcData structure is defined in omtcdata.h. The structure defines the raw 64 bits of the timecode as defined in the standard "SMPTE 12M-1999, Time and Control Code". The structure uses bit definitions; it has been carefully designed to fit into 64 bits. See omtcdata.h for more information.

Enum OmMediaClipDataType

Description

Describes the type of clip data to be used.

```
enum OmMediaClipDataType {
omClipDataUnknown,
    omClipDataCcField1,
    omClipDataCcField2,
    omClipDataCcFrame,
    omClipDataTimecode,
```

```
                omClipDataUserbits,
                omClipDataTimeUser,
                omClipDataOmTcData,
                omClipDataRawVbi,
            omClipData436Vbi,
            omClipData436Vanc,
            omClipDataVideoEmbVanc
};
```

Descriptions of the *OmMediaClipDataType* enum fields are provided in the following table.

OmMediaClipDataType Enum Fields	Description
omClipDataUnknown	Invalid
omClipDataCcField1	2 bytes of closed caption from field 1
omClipDataCcField2	2 bytes of closed caption from field 2
omClipDataCcFrame	4 bytes of closed caption
omClipDataTimecode	4 bytes of timecode (8 TC nibbles of OmTcData)
omClipDataUserbits	4 bytes of userbits (8 UB nibbles of OmTcData)
omClipDataTimeUser	4 bytes of timecode followed by 4 bytes userbits
omClipDataOmTcData	8 bytes in OmTcData format
omClipDataRawVbi	OmPlrRawVbiHeader followed by 1440 bytes per line
omClipData436Vbi	SMPTE 436M Wrapped VBI
omClipData436Vanc	SMPTE 436M Wrapped VANC
omClipDataVideoEmbVanc	VANC stored in Video Stream (e.g. MPEG Smpte328m)

Remarks

None to note.

Enum OmMediaCopyType

Description

Describes the type of copy to be performed.

```
enum OmMediaCopyType {
OmFlattenedWithDiscreteMedia
OmFlattenedWithEmbeddedMedia
OmReferenceCopy
};
```

Descriptions of the *OmMediaCopyType* enum fields are provided in the following table.

OmMediaCopyType Enum Fields	Description
omFlattenedWithDiscreteMedia	Default value if not specified. All essence media will be copied (flattened) to new, discrete files in a directory named media.dir beneath the location of the destination movie. The essence file names will be identical to the movie, except for the suffix.
omFlattenedWithEmbeddedMedia	All essence media will be copied (flattened) to the destination movie, being embedded inside it. The new movie is, in effect, self-contained.
omReferenceCopy	No essence media is copied but is left in place. However, the new destination movie is constructed with references to these essence files. In this case, it is a good idea to make sure the essence files are located in the same directory as, or below, the destination movie file.

Remarks

None to note.

Enum OmMediaFileType

Describes the file type for file readers and writers.

```
enum OmMediaFileType {
omMediaFileTypeUnknown = 0,
omMediaFileTypeDV,
omMediaFileTypeQt,
omMediaFileTypeMxf,
omMediaFileTypeMpegsTs,
omMediaFileTypeMpegsPs,
omMediaFileTypeMpeg4,
omMediaFileTypeMpegV,
omMediaFileTypeMpegA1,
omMediaFileTypeMpegA2,
omMediaFileTypeMpegA3,
omMediaFileTypeHdcam,
omMediaFileTypeRec601,
omMediaFileTypeAiff,
omMediaFileTypeWave,
omMediaFileTypeAc3,
omMediaFileTypeVbi,
omMediaFileTypeData,
omMediaFileTypeAes3Audio,
```

```
    omMediaFileTypeNull,

    OmMediaFileTypes
    };
```

Descriptions of the *OmMediaFileType* enum fields are provided in the following table.

File Types	OmMediaFileType Enum Fields	Description
Containers	omMediaFileTypeDv	IEC-61834 DV video/audio
	omMediaFileTypeQt	QuickTime container file
	omMediaFileTypeMxf	MXF container file
	omMediaFileTypeMpegTs	MPEG transport stream
	omMediaFileTypeMpegPs	MPEG program stream
	omMediaFileTypeMpeg4	MPEG-4 container

File Types	OmMediaFileType Enum Fields	Description
Single Track Elementary Files	omMediaFileTypeMpegV	MPEG video
	omMediaFileTypeMpegA1	MPEG 1 audio, layer 1
	omMediaFileTypeMpegA2	MPEG 1 audio, layer 2
	omMediaFileTypeMpegA3	MPEG 1 audio, layer 3
	omMediaFileTypeHdcam	Sony HDCAM
	omMediaFileTypeRec601	Uncompressed ITU rec 601; aka CCIR 601
	omMediaFileTypeAiff	Apple's AIFF
	omMediaFileTypeWave	Microsoft's WAV
	omMediaFileTypeAc3	Dolby's AC3
	omMediaFileTypeVbi	Omneon proprietary VBI
	omMediaFileTypeData	Omneon proprietary data stream
	omMediaFileTypeAes3Audio	Raw PCM audio data
	omMediaFileTypeNull	
	OmMediaFileTypes	Must be last

Remarks

None to note.

Enum OmVideoFormat

Description

Describes the video format to be used.

```
enum OmVideoFormat {
omVidFmtInvalid,
omVidFmt525Line29_97Hz2_1,
omVidFmt625Line25Hz2_1,
omVidFmt1125Line29_97Hz2_1,
omVidFmt1125Line25Hz2_1,
```

```
    omVidFmt750Line59_94Hz,
    omVidFmt750Line50Hz,
    };
```

Descriptions of the *OmVideoFormat* enum fields are provided in the following table.

OmVideoFormat Enum Fields	Description
omVidFmtInvalid	Invalid value
omVidFmt525Line29_97Hz2_1	525 lines/frame, 29.97 Hz frame rate, 2:1 interlace
omVidFmt625Line25Hz2_1	625 lines/frame, 25 Hz frame rate, 2:1 interlace
omVidFmt1125Line29_97Hz2_1	SMPTE 274M System 5: 1125 lines/frame (1080 active); 2200 (1920 active), 30/1.001 Hz frame rate, 74.25/1.001 MHz clock, 2:1 interlace
omVidFmt1125Line25Hz2_1	SMPTE 274M System 6: 1125 lines/frame (1080 active), 2640 (1920 active), 25 Hz frame rate, 74.25 MHz clock, 2:1 interlace
omVidFmt750Line59_94Hz	750 lines/frame, 59_94 Hz frame rate
omVidFmt750Line50Hz	750 lines/frame, 50 Hz frame rate

Remarks

None to note.

Enum OmVideoScan

Description

Describes the video field scan used.

```
enum OmVideoScan {
    omVidScanInvalid,
    omVidScan1080Line1920Pixel,
    omVidScan720Line1280Pixel,
    omVidScan480Line720Pixel,
    omVIdScan576Line720Pixel
};
```

Descriptions of the *OmVideoScan* enums fields are provided in the following table.

OmVideoScan Enum Fields	Description
omVidScanInvalid	omVidScanInvalid
omVidScan1080Line1920Pixel	SMPTE 274M: 1080 active lines, 1920 active pixels
omVidScan720Line1280Pixel	SMPTE 296M: 720 active lines, 1280 active pixels

OmVideoScan Enum Fields	Description
omVidScan480Line720Pixel	CCIR 601, 480 active lines, 720 active pixels
omVidScan576Line720Pixel	CCIR 601, 576 active lines, 720 active pixels

Remarks

None to note.

Enum OmFrameStruct

Description

Describes the video field structure used.

```
enum OmFrameStruct {
    omFrmStructInvalid,
    omFrmStructInterlace,
    omFrmStructProgressive
};
```

Descriptions of the *OmFrameStruct* enums fields are provided in the following table.

OmFrameStruct Enum Fields	Description
omFrmStructInvalid	Invalid frame structure
omFrmStructInterlace	Interlaced frame
omFrmStructProgressive	Progressive frame

Remarks

None to note.

Enum OmFieldRate

Describes the video field rate used.

```
enum OmFieldRate {
    omFldRateInvalid,
    omFldRate24Hz,
    omFldRate50Hz,
    omFldRate59_94Hz,
    omFldRate60Hz
};
```

Descriptions of the *OmFieldRate* enums fields are provided in the following table.

OmFieldRate Enum Fields	Description
omFldRateInvalid	Invalid rate
omFldRate24Hz	Rate is 24Hz

OmFieldRate Enum Fields	Description
omFldRate50Hz	Rate is 50Hz
omFldRate59_94Hz	Rate is 59.94Hz
omFldRate60Hz	Rate is 60Hz

Remarks

None to note.

Enum OmStreamType

Description

Describes the top level stream (wire) containers.

```
enum OmStreamType {
omStreamUnknown = 0,
omStreamMpegVes = 1,
omStreamPcmAudio = 5,
omStreamDv = 6,
omStreamRec601Video = 8,
omStreamHdcam = 9,
omStreamVbi = 10,
omStreamMuxed = 11,
omStreamData = 12,
omStreamMpegTs = 13,
omStreamMxf = 14,
omStreamTc = 15
};
```

Descriptions of the *OmStreamType* enums fields are provided in the following table.

OmStreamType Enum Fields	Description
omStreamMpegVes = 1,	MPEG video elementary stream
omStream... = 2 to 4	Unused
omStreamPcmAudio = 5,	PCM audio
omStreamDv = 6,	DV-style DIF blocks
omStream... = 7	Unused
omStreamRec601Video = 8	VBI stream

OmStreamType Enum Fields	Description
omStreamHdcam = 9	Unused
omStreamVbi = 10	Unused
omStreamMuxed = 11	Multiplexed video/audio
omStreamData = 12	Streamed data (unknown type)
omStreamMpegTs = 13	MPEG transport stream
omStreamMxf = 14	MXF stream (future implementation)
omStreamTc = 15	Timecode

Remarks

Similar to *OmMediaType*, but *OmStreamType* reflects the top-level container on the wire, while *OmMediaType* represents individual data types. Currently most media types appear on the wire as themselves, without containers.

Enum OmFrameRate

Description

Describes the video frame rate used.

```
enum OmFrameRate {
omFrmRateInvalid,
omFrmRate24Hz,
omFrmRate25Hz,
omFrmRate29_97Hz,
omFrmRate30Hz,
omFrmRate50Hz,
omFrmRate59_94Hz,
};
```

Descriptions of the *OmFrameRate* enum fields are provided in the following table.

OmFrameRate Enum Fields	Description
omFrmRateInvalid	Invalid video frame rate
omFrmRate24Hz	Rate is 24Hz
omFrmRate25Hz	Rate is 25Hz
omFrmRate29_97Hz	Rate is 29.97Hz

OmFrameRate Enum Fields	Description
omFrmRate30Hz	Rate is 30Hz
omFrmRate50Hz	Rate is 50Hz
omFrmRate50Hz	Rate is 59.94Hz

Remarks

None to note.

Enum OmFieldID

Describes the video field ID used.

```
enum OmFieldId {
omFieldInvalid,
omFieldAny = omFieldInvalid,
omField1,
omField2
};
```

Descriptions of the *OmFieldID* enum fields are provided in the following table.

OmField Enum Fields	Description
omFieldInvalid	Invalid video field ID
omFieldAny = omFieldInvalid	Invalid video field ID
omField1	Identifies field as field 1
omField2	Identifies field as field 2

Remarks

None to note.

Enum OmMediaType

Description

Describes the basic type of media contained in a single track of a clip.

```
enum OmMediaType {
omMediaUnknown      = 0,
omMediaMpegVideo    = 1,
omMediaMpegStdAudio = 2,
omMediaMpegAc3      = 3,
omMediaPcmAudio     = 5,
omMediaDvVideo      = 6,
omMediaDvAudio      = 7,
omMediaRec601Video  = 8,
omMediaHdcam        = 9,
```

```
    omMediaVbi          =10,
    omMediaData         =12,
    omMediaNumTypes     =13
    };
```

Descriptions of the *OmMediaType* enum fields are provided in the following table.

OmMediaType Enum Fields	Description
omMediaUnknown	Invalid media type
omMediaMpegVideo	Standard, IEC-13818 video
omMediaMpegStdAudio	MPEG layer 1, 2, 3 audio
omMediaMpegAc3	Dolby AC3 audio in an MTS
omMediaPcmAudio	PCM audio
omMediaDvVideo	DV video
omMediaDvAudio	DV audio
omMediaRec601Video	Standard SMPTE Rec 601 video
omMediaHdcam	HDCAM video
omMediaVbi	Vertical blanking interval
omMediaData	Update OmMediaNumTypes
OmMediaNumTypes	Number of types in OmMediaType

Remarks

None to note.

Enum {OmMediaNumTypes = 13}

Description

Lists the number of media types inOmMediaType.

```
enum {OmMediaNumTypes = 13};
```

There are no fields associated with this enum.

Remarks

None to note.

Enum OmVideoSampleRatio

Description

Describes the picture sample ratio among luma and chroma.

```
enum OmVideoSampleRatio {
    omVideoSampleNone,
```

```
        omVideoSample420,
        omVideoSample411,
        omVideoSample422,
        omVideoSample444
    };
```

Descriptions of *OmVideoSampleRatio* enum fields are provided in the following table.

OmVideoSampleRatio Enum Fields	Description
omVideoSampleNone	Non-video data
omVideoSample420	4:2:0 data
omVideoSample411	4:1:1 data
omVideoSample422	4:2:2 data
omVideoSample444	4:4:4 data

Remarks

None to note.

Enum OmVideoAspectRatio

Description

Describes the picture aspect ratio.

```
        enum OmVideoAspectRatio {
        omVideoAspectNone,
        omVideoAspect4to3,
        omVideoAspect16to9,
    };
```

Descriptions of the *OmVideoAspectRatio* enum fields are provided in the following table.

OmVideoAspectRatio Enums Fields	Description
omVideoAspectNone	Non-video data
omVideoAspect4to3	4:3 picture aspect ratio
omVideoAspect16to9	16:9 picture aspect ratio

Remarks

None to note.

Enum OmAudioFormat

Description

Describes how audio is encoded.

```
enum OmAudioFormat {
    omAudioFormatUnknown = 0,
    omAudioFormatPcm = 1,
    omAudioFormatRaw = 2,
    omAudioFormatAC3 = 3,
};
```

Descriptions of the *OmAudioFormat* enum fields are provided in the following table.

OmAudioFormat Enum Fields	Description
`omAudioFormatUnknown = 0`	Unknown encoding format
`omAudioFormatPcm = 1`	PCM audio format
`omAudioFormatRaw = 2`	Raw audio format
`omAudioFormatAC3 = 3`	AC3 audio format

Remarks

None to note.

Enum OmMediaPropertyType

Description

Describes the property type for the media.

```
enum OmMediaPropertyType {
omMediaPropertyString,
omMediaPropertyPrivate,
omMediaPropertyuint8,
omMediaPropertyuint16,
omMediaPropertyuint32,
omMediaPropertyuint64,
omMediaPropertyint8,
omMediaPropertyint16,
omMediaPropertyint32,
omMediaPropertyint64,
omMediaPropertyfloat32,
omMediaPropertyfloat64,
omMediaPropertyBinary,
};
```

Descriptions of the *OmMediaPropertyType* enum fields are provided in the following table.

OmMediaPropertyType Enum Fields	Description
`omMediaPropertyString`	Property is a text string
`omMediaPropertyPrivate`	Property is a user-defined private structure
`omMediaPropertyuint8`	Property is uint8

OmMediaPropertyType Enum Fields	Description
omMediaPropertyuint16	Property is uint16
omMediaPropertyuint32	Property is uin32
omMediaPropertyuint64	Property is uint64
omMediaPropertyint8	Property is int8
omMediaPropertyint16	Property is int16
omMediaPropertyint32	Property is int32
omMediaPropertyint64	Property is int64
omMediaPropertyfloat32	Property is a 32-bit float
omMediaPropertyfloat64	Property is a 64-bit float
omMediaPropertyBinary	Property is non-specific binary

Remarks

None to note.

Appendix A
Glossary

Abbreviations and Terms Used in Omneon Media API

Abbreviation/Term	Explanation
.AIFF	Audio Interchange File Format
.DV	Digital Video
.MXF	Material eXchange Format
.WAV	Waveform Audio File Format
Clip	Describes a piece of media that is a permanent object stored in the Omneon file system, or a piece of media that is a temporary object attached to a Timeline associated with an Omneon Player.
Essence	Essence refers to elementary media files.
FCP	(Apple) Final Cut Pro
GOP	Group of Pictures
ITU	International Telecommunications Union
Mbps	Megabits per second
Media File	Refers to elementary media, such as audio or video. This API treats both audio and video media in an identical manner, so that while one file may contain audio and video another might contain just audio.
Movie	A file composed of both audio and video and is frequently is associated with Apple's QuickTime files.
MPEG	Motion Picture Experts Group
MSC	Media Storage Chassis
NAS	Network Attached Storage
Player	A software object inside the Omneon MediaDirector that is responsible for pushing the media out onto the IEEE 1394 based network that connects the MediaDirector to Omneon MediaPort. The Player broadcasts the media out onto the bus and MediaPorts are configured to listen for the broadcast.
Timeline	A software object used to play out lists of clips.
VBI	Vertical Blanking Interval

Appendix B
Notes about the Media API

This appendix provides answers to the most frequently asked questions relating to the Media API in the following categories:

- *About File Access*
- *About File and Wrapper Formats used by Spectrum*
- *Media API Support for and Integration with Third Party Applications*
- *Notes on Renaming Files in Media API*

About File Access

In designing your application, it is helpful to understand how the Media API accesses files. The Media API has two main groups of functions, those that deal with moving clips from one place to another (**OmMediaCopier**) and those that provide clip management and read/write access to clip metadata (**OmMediaQuery**). Note the following differences in the way the two groups access files.

OmMediaQuery File Access

OmMediaQuery functions can access any file available through the local file system. However, note the following restrictions:

- To use OmMediaQuery functions to access files stored on a MediaDirector, the client PC on which your application runs must support the Common Internet File System (CIFS) client protocol. On Windows*, CIFS is the native protocol used by Windows File Sharing. On Linux, CIFS is supported by Samba.
- To use OmMediaQuery functions to access files stored on an Omneon MediaGrid, the Omneon MediaGrid File System Driver (FSD) must be installed on the client PC. For instructions on installing the MediaGrid FSD, refer to the *Omneon MediaGrid Installation and Configuration Guide*.

This open design enables your application to use the local file system to access media files regardless of where they are stored. If OmMediaQuery can access a file, then your application can as well. Your application may move, copy and rename files, but is responsible for maintaining the relative relationships between media wrappers and essence files, and for using correct file suffixes.

OmMediaCopier File Access

OmMediaCopier functions access files in two ways, using the “local” mode or the “remote” mode. The remote mode is enabled by the use of the setRemoteHost function.

- When using the local mode, files are accessed through the local file system in the same way as the OmMediaQuery functions.
- When using the remote mode, all the data intensive tasks are offloaded to a remote system with the client PC only functioning as a control agent. The remote mode of OmMediaCopier uses the Sun Remote Procedure Call (RPC) protocol to communicate with the remote host. At this time, the ContentBridge component of the Omneon MediaGrid and the Spectrum MediaDirector can both be used as remote hosts.

NOTE: When using remote mode, the fact that OmMediaCopier can access a remote file does not mean that your application will be able to do the same.

NOTE: If necessary, you can use an open source software utility, such as ONC rpcinfo, to determine the IP port numbers used for RPC connections.

The following illustration shows how the two groups of functions access files on Spectrum and Omneon MediaGrid systems. Note that in order to access files on the Omneon MediaGrid without a ContentBridge, you must have the necessary file system driver (FSD) installed.

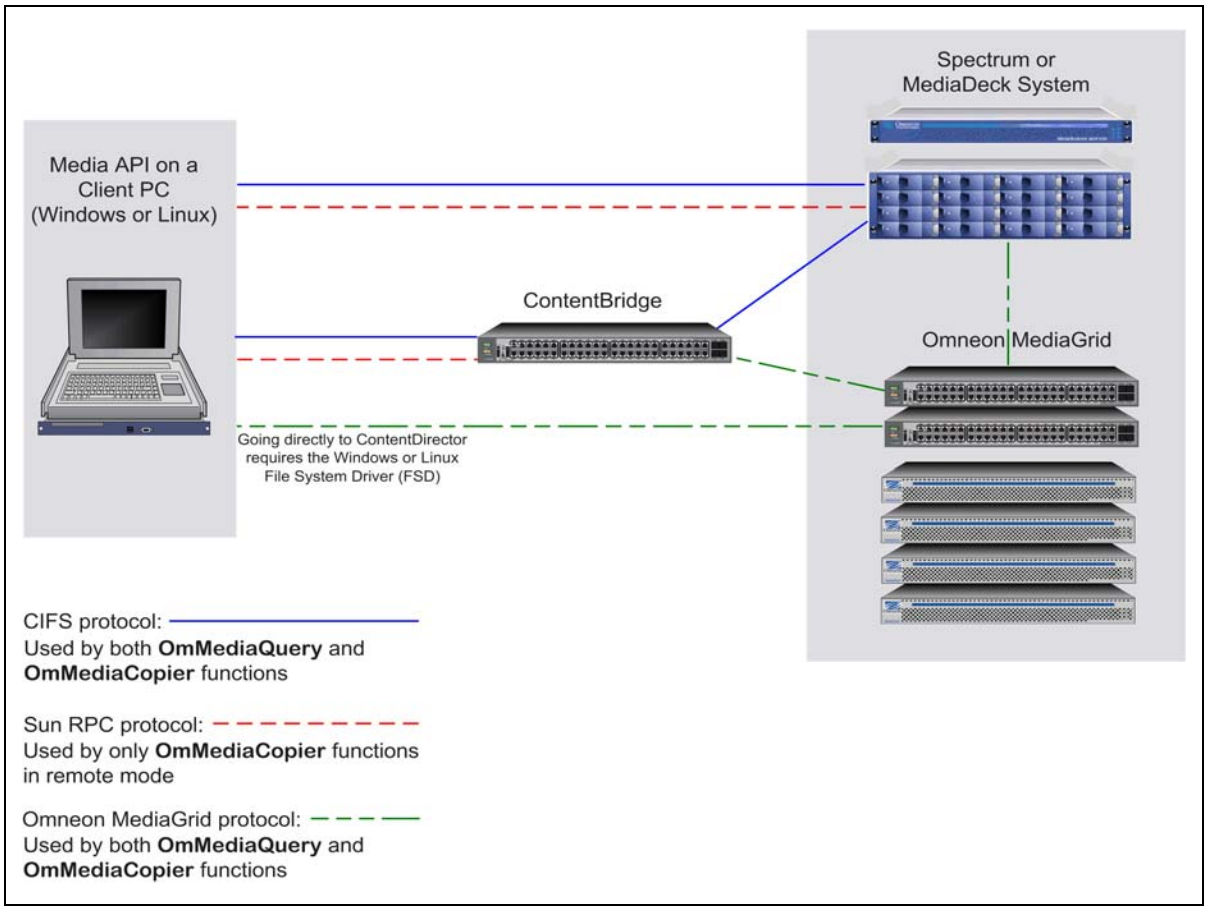

Figure 0–1: Accessing Files on Spectrum or Omneon MediaGrid system

NOTE: When using OmMediaCopier, always copy from and to a subdirectory of the file system and not the root directory. Copying from or to the root directory may result in a failed copy.

About File and Wrapper Formats used by Spectrum

Refer to “Spectrum Media and Wrapper Formats” for combinations of supported video and wrapper formats.

Media API Support for and Integration with Third Party Applications

This section provides information on the following topics:

- *Limitations on References to Material Files and Jumps in QuickTime Files*
- *Notes on Parsing QuickTime Files*
- *Constructing QuickTime Movies from Scratch for Export to an Spectrum MediaDirector*

Limitations on References to Material Files and Jumps in QuickTime Files

In general, there are no constraints on the edits expressed within the QuickTime file. However, some technical limitations exist for each opened media files including DV media files stored in “large stripe” files and audio files stored in “small stripe” files.

Smallstripe files are those that contain media of relatively low bitrates for realtime playout. They use a larger block size on disk and consume a medium amount of MediaDirector resources. On an Spectrum System, .aiff and .wav audio files fall into this “smallstripe” category.

Largestripe files are those that contain high bitrate realtime media. They use the largest block size on disk and consume a large amount of MediaDirector resources. On an Spectrum system, all video files (including .dv, .601, .mpg etc.) fall into this “largestripe” category.

The following limitations apply:

- If a player on the MediaDirector opens and reads as little as a single byte from a DV media file or an audio file, the file system cache will consume one stripe's worth of memory as long as the file descriptor remains open. A large stripe is 256KB times the number of non-redundant disks in a raid set, and the multiplier for a large stripe is 64KB. For example, with an 8+1 raid set, a large stripe cache entry would be 8*256k=2MB.
- During playback, a player will cache 60 frames into the future. Thus, if you have a movie with 60 1-frame edits where the media resides in separate files, you may end up consuming 120MB of file system cache. Since memory allocation for the DV channel is limited to 10MB in the Spectrum System software release 1.4, this will cause the system to fail. Using software release 2.0 or later, the system should not fail due to a change in allocation policy. However, physical limits on the memory may mean that other players or recorders will not work correctly.

NOTE: A mechanism exists in the QuickTime code that closes files from previous edits thus improving memory usage. Refer to Apple's QuickTime documentation for more information.

- Even if the media files are all exactly one frame long, the file system allocation is such that reading a single frame will result in a full stripe's worth of I/O (2MB for DV on an 8+1 raid set). Thus, a movie composed entirely of one-frame edits and media files requires about 30fps*2MB per frame or 60MB per second. Since this is close to the I/O capacity for a single raid set, other players/recorders may be impacted.

Notes on Parsing QuickTime Files

The parsing of QuickTime files should only be attempted by those who need an independent code base that can handle the QuickTime file format. The following notes are intended to address QuickTime file format parsing in general, and specifically, the “alis” atom, which is used to hold pointers to external media files:

- You can download Apple's QuickTime player for free, and upgrade it to the pro version at a low cost. If you intend to create QuickTime files, you can use it to confirm that they were properly constructed. If you intend to read QuickTime files, you can use it to confirm resolution of externally referenced media files. An excellent method to test your code is to use the QuickTime player, create your own movie from a separate video and audio file, save the movie, and then try to parse the file. If you cannot parse it, then your code does not match Apple's.
- The Apple QuickTime file format specification is well written. Spectrum always strives to maintain compatibility with Apple's definition of QuickTime, so in general, if Apple's software can understand the QuickTime files produced by the QuickTime player or Final Cut Pro, then MediaDirectors should understand it as well. If this is not the case, contact Harmonic Technical Support for assistance. You can also use Apple's Dumpster to look at the low-level structure of a QuickTime file. If you are working on a PC or an Apple platform, consider using Apple's QuickTime API to read and write QuickTime files.
- Apple's open source code for the QuickTime streaming server parses QuickTime files, including the resolution of “alis” atoms. Apple's streaming server source code is available at: *http://developer.apple.com/darwin/projects/streaming*. Go to QTAtom_dref::ResolveAlias() as a starting point.
- Since the correct tools are important, especially when trying to decipher binary files, Harmonic strongly suggest downloading a free copy of Bvi from: **http://bvi.sourceforge.net/**

This allow you to more easily see where the atoms are located and see the encoding of the “pascal strings” used in QuickTime files. If you prefer “od”, you can get an up-to-date version that provides both a binary and text view from GNU here: **http://www.gnu.org/directory/GNU/textutils.html**.

Constructing QuickTime Movies from Scratch for Export to an Spectrum MediaDirector

Choose from the following methods to construct QuickTime movies from scratch for export to an MediaDirector:

- Use the standard Apple QuickTime library. Note however that Harmonic and Apple have differing implementations of MPEG, HDCAM, and DATA inside QuickTime, and the two implementations are not compatible. In addition, the QuickTime library is intended as an end-to-end solution for those who do everything with QuickTime, not as a utility to export a file format.
- Write your own QuickTime file format parser/exporter. This is not an easy task. It is possible to write a simple parser in a day or so, but it can take considerable time to write one that can properly interpret all the QT atoms associated with a multi-track movie. In addition, writing the code to create a QT file takes even more time, especially if you attempt to tackle more subtle variations like edits and multiple media files in a single track.

Notes on Renaming Files in Media API

Keep the following in mind:

- When a clip is renamed within the same directory (that is "mvclip clipa clipb"), only the movie file is renamed.
- When a clip rename crosses a directory (that is "mvclip /fs1/clip.dir/clipa.mov /fs1/private/clipb.mov), the media files will be renamed.

These behaviors occur because if media files are renamed, the QuickTime file must be rewritten also. This is a potentially time consuming process compared to a file rename which can be almost instantaneous.

Appendix C
Omneon Neutral VANC Format

This appendix provides informaton about the Omneon neutral vertical ancillary data (VANC) format. Choose from the following topics:

- *About Omneon Neutral VANC Format*
- *Frame Header Format*
- *Omneon VANC Packet*
- *Modified VANC Payload*
- *VANC Usage Notes*

About Omneon Neutral VANC Format

Omneon has defined a neutral VANC format for use in transcoding between the Media API supported VANC formats. This format allows an API user to encode and decode a single VANC format to/from the Media API while giving the API user access to the various Ancillary Data formats. The supported VANC formats are:

- MPEG-2 SMPTE 328M
- DVCPro100 SMPTE 375M
- AVC-Intra Panasonic
- MXF SMPTE 436M

The Neutral VANC stream is structured as shown in *Figure 0–2*. A VANC Frame Header precedes each group of VANC Packets for a given frame. Each VANC Payload is preceded by a VANC Packet Header. Each VANC Payload easily maps to a SMPTE 291 VANC Packet.

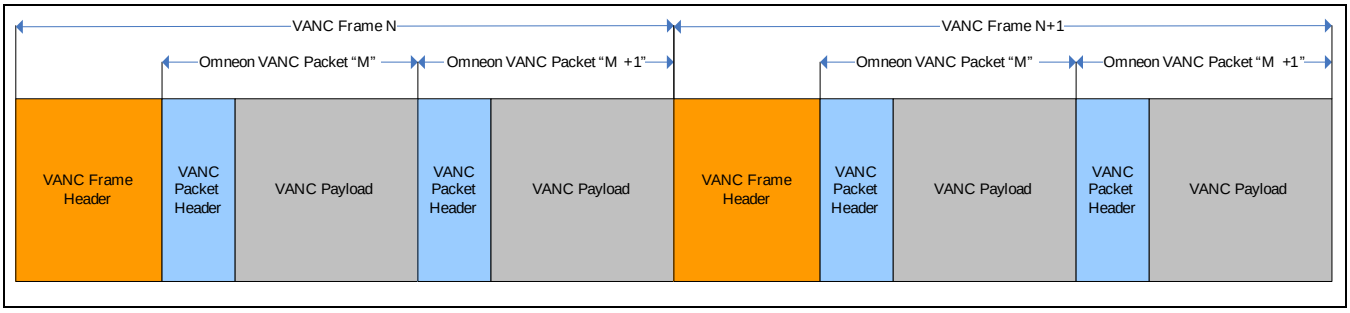

Figure 0–2: Omneon Neutral VANC Stream

Frame Header Format

Currently Omneon supports two distinct Frame Header formats. The Media API during insertion operations will accept either format. During extraction operations the Media API defaults to Type-2 Omneon VANC Frame Headers.

Omneon Frame Header Type-1

Type-1 frame headers are not available by default when extracting data. Type-1 Frame Headers are accepted by the Media API during insertion operations. To enable Type-1 headers use during extraction, use the Media API debug clipDataDebug and 4-byte frame headers.

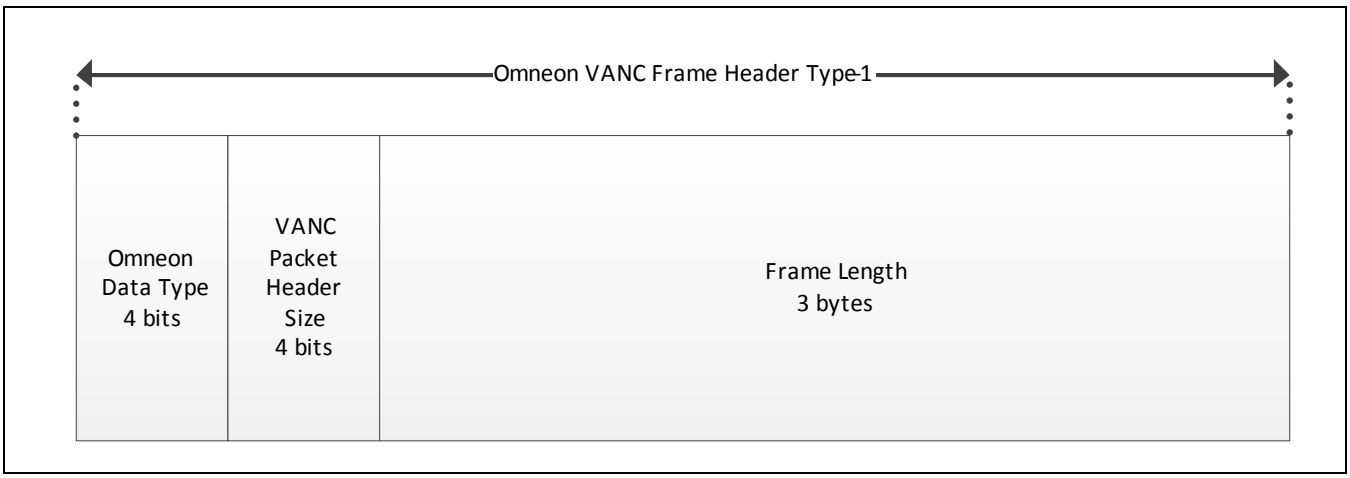

Figure 0–3: Frame Header Type-1

Table 0–1:

Field	Length	Description
Omneon Data Type	4 bits	Defines the frame header type: 0x01: Type 1 0x02: Type-2 0x03: reserved … 0x0f: reserved
VANC Packet Header Size	4 bits	Size of each VANC Packet Header in bytes
Frame Length	3 bytes	Length of this Frame in bytes including the Frame Header

Omneon Data Type

Defines the type of Omneon Frame header. For Type-1 Headers the value is set to 0x1.

VANC Packet Header Size

The size of the Omneon VANC Packet Header in bytes.

Frame Length

The length of the entire Omneon VANC Frame including the Frame Header.

Frame Header + Omneon VANC Packet 0 + Omneon VANC Packet 1 + … + Omneon VANC Packet N)

Omneon Frame Header Type-2

Type-2 frame headers are used by default by the Media API. They may be used during extraction and insertion.

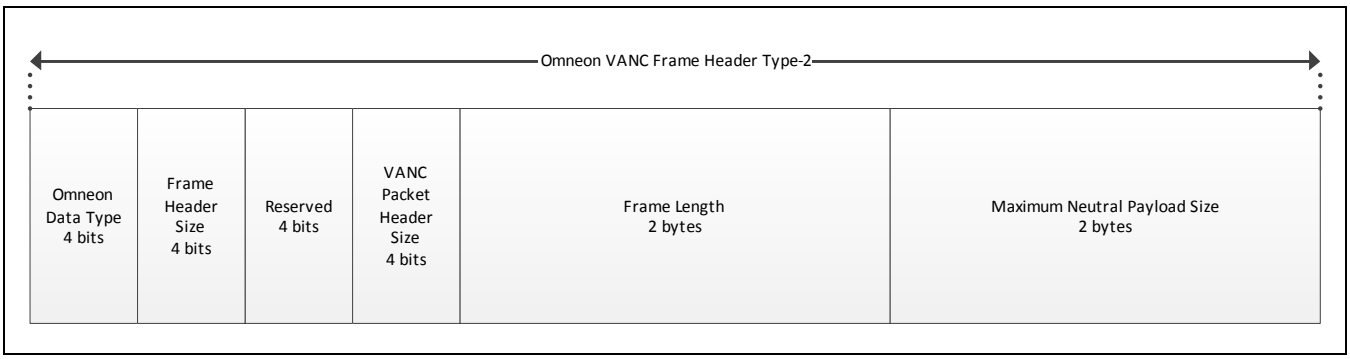

Figure 0–4: Frame Header Type-2

Table 0–2:

Field	Length	Description
Omneon Data Type	4 bits	Defines the frame header type: 0x01: Type 1 0x02: Type-2 0x03: reserved … 0x0f: reserved
Frame Header Size	4 bits	Size of the Frame Header in bytes
Reserved	4 bits	Reserved, set to zero (0x00)
VANC Packet Header Size	4 bits	Size of each VANC Packet Header in bytes
Frame Length	2 bytes	Length of this Frame in bytes including the Frame Header
Maximum Payload Size	2 bytes	Space available in bytes for ancillary data in this frame

Omneon Data Type

Defines the type of Omneon Frame Header. For Type-2 Headers the value is set to 0x02.

Frame Header Size

The size of the Frame Header in bytes. For Type-2 the value is set to 0x06.

VANC Packet Header Size

The size of the Omneon VANC Packet Header in bytes.

Frame Length

The length of the entire Omneon VANC Frame including the Frame Header.

Frame Header + Omneon VANC Packet 0 + Omneon VANC Packet 1 + … + Omneon VANC Packet N)

Max Neutral Payload Size

The maximum Omneon Neutral Payload size (excludes Frame Header) that will fit in the current frame. This value is a calculated worst-case VANC encoding scenario, where the data is broken up into as many minimally-sized VANC packets as possible (i.e. most of the space is taken up by header data). This field has no meaning when inserting data.

Omneon VANC Packet

The Omneon VANC packet is made up of an Omneon VANC packet header and an Omneon VANC payload. Those structures are described in more detail below.

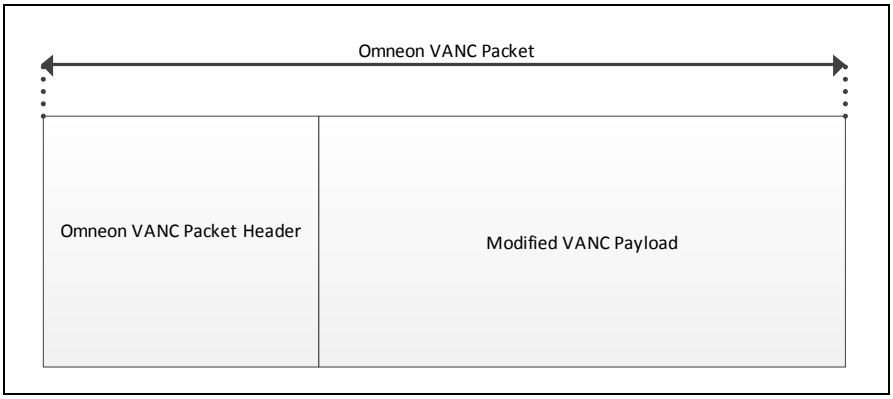

Figure 0–5: Omneon VANC Packet

Omneon VANC Packet Header

The VANC Packet Header is structured as shown in *Figure 0–6*.

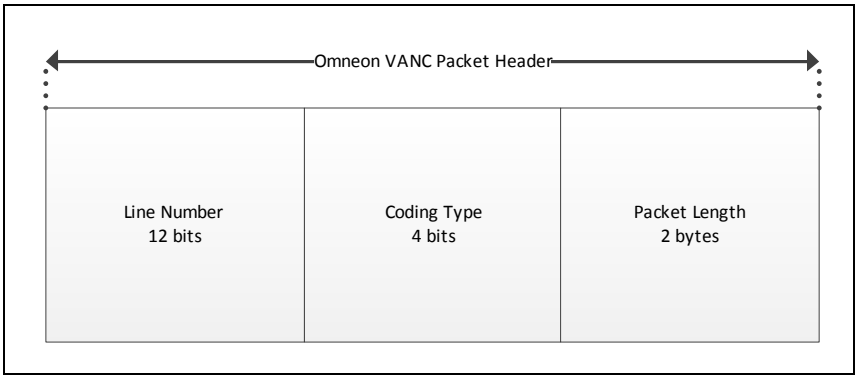

Figure 0–6: Omneon VANC Packet Header

Table 0–3:

Field	Length	Description
Line Number	12 bits	Line number of Data Values of 1 to N (where N is the number of lines in a frame) 0 = no available line data
Coding Type	4 bits	VANC Encoding Parameters 0 – Reserved 1 – 8-bit, Luma, Frame Wrapped (Default) 2 – Reserved 3 – Reserved …. 15 – Reserved
Packet Length	16 bits	Length in bytes of entire VANC Packet including header (packet header + VANC Payload)

- **Line Number**: The associated line number with the VANC packet as a value of 1 to N, where N is the number of lines in a frame. The value of 0 indicates that no line data is available, but is not a valid value when writing VANC data with the API.
- **Coding Type**: Specifies the VANC Payload encoding format.
- **Packet Length**: Specifies the length of the entire VANC Packet including both the VANC Packet Header and the Modified VANC Payload.

Modified VANC Payload

The VANC payload in an Omneon VANC packet is a modified SMPTE 291M VANC Packet shown in *Figure 0–7*.

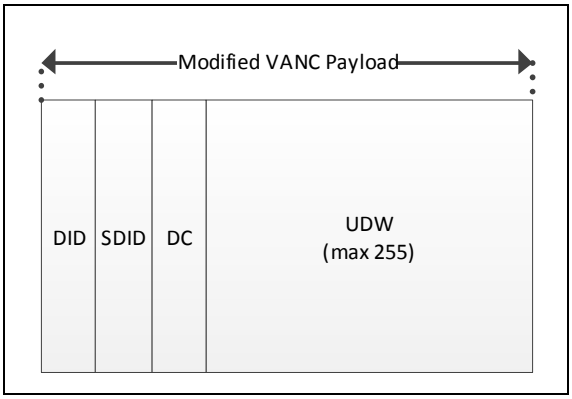

Figure 0–7: Modified VANC Payload

Figure 0–8 shows the mapping of an Ancillary Packet to the Modified VANC Packet.

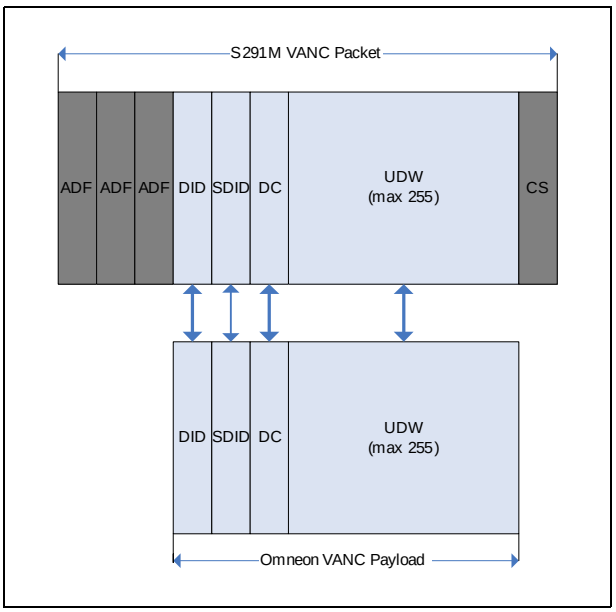

Figure 0–8: Mapping of ANC Packet to VANC Payload

VANC Usage Notes

- Can the MAPI add VANC to a file that has no existing VANC?

The MAPI can only add VANC to a file that has existing space for VANC.

- What does the MAPI do with existing VANC in a frame when writing new VANC?

The MAPI throws away existing VANC. The recommended order of operations is:

i. Extract VANC.
ii. Perform VANC Processing in your application.
iii. Insert VANC from step 2.

- ATSC Packets

You cannot extract just an ATSC packet. You need to extract the entire VANC payload from a frame, then you need to search the payload for the ATSC/53 packet.

Appendix D
Contacting the Technical Assistance Center

Harmonic Global Service and Support has many Technical Assistance Centers (TAC) located Globally but virtually co-located where our customers can obtain technical assistance or request on-site visits from the Regional Field Service Management team. The TAC operates a Follow-The-Sun support model to provide Global Technical Support anytime, anywhere, through a single case management and virtual telephone system. Depending on time of day, anywhere in the world, we will receive and address your calls or emails in one of our global support centers. The Follow-the-Sun model greatly benefits our customers by provided continuous problem resolution and escalation of issues around the clock.

Table D–1: For Distribution and Delivery (Legacy Harmonic) Products

Region	Telephone Technical Support	E-mail
Americas	888.673.4896 or 408.490.6477	*support@harmonicinc.com*
EME	+44.1252.555.450	*support.emea@harmonicinc.com*
Asia Pacific – Other Territories	+852.3713.9300	*hongkongtechsupport@harmonicinc.com*
India	+44.1252.555.450	*support.emea@harmonicinc.com*
Russia	+7.495.926.4608	*rusupport@harmonicinc.com*
Africa	+44.1252.555.450	*support.emea@harmonicinc.com*
Mainland China	+86.10.8391.3313	*chinatechsupport@harmonicinc.com*

Table D–2: For Production and Playout (Legacy Omneon and Rhozet) Products

Region	Telephone Technical Support	E-mail
Americas	888.673.4896 or 408.490.6477	*omneon.support@harmonicinc.com*
EMEA	+44.1252.555.450	*omneonemeasupport@harmonicinc.com*
Asia Pacific – Other Territories	+65.6542.0050	*apacsupport@harmonicinc.com*
Japan	+81.3.5565.6737	*japansupport@harmonicinc.com*
China - Mainland	+86.10.8391.3313	*chinasupport@harmonicinc.com*
Russia and CIS	+7.495.926.4608	*rusupport@harmonicinc.com*

The Harmonic Inc. support website is:

http://www.harmonicinc.com/content/technical-support

The Harmonic Inc. Distribution and Delivery product software downloads site is:

ftp://ftp.harmonicinc.com

The Harmonic Inc. Playout and Production software downloads site is:

ftp://ftp.Omneon.com//Updates/Omneon/Current/

The Harmonic Inc. corporate address is:

Harmonic Inc.
4300 North First St.
San Jose, CA 95134, U.S.A.
Attn: Customer Support

The corporate telephone numbers for Harmonic Inc. are:

Tel. 1.800.788.1330 (from the U.S. and Canada)
Tel. +1.408.542.2500 (outside the U.S. and Canada)
Fax.+1.408.542.2511

HEADQUARTERS
Americas Sales
4300 North First Street
San Jose, CA 95134 U.S.A.
T 1 800 828 5521 inside the U.S.
+1 408 542 2559 outside the U.S.
F +1 408 490 6001

ASIA-PACIFIC
Harmonic (Asia Pacific) Limited
Suite 2301, L23, Office Tower
Langham Place, 8 Argyle St
Mongkok, Kowloon Hong Kong
T +852 3713 9300
F +852 2116 0083

EUROPE AND MIDDLE EAST
United Kingdom
250 Fowler Avenue, Ground Floor
IQ Farnborough
Farnborough Hampshire GU14 7JP
United Kingdom
T +44 (0)1 252 555 400
F +44 (0)1 252 377 171

Africa, India, Russia and CIS Countries
10 Haamel St
Park Afek
Rosh Ha'ayin, 48092 Israel
T +972.3.9007777
+972.3.9007800
F +972.3.9007766