Spectrum™ Player API

Release 8.2

Programmer Guide

Rev A

Manual Part No. 28-0036
September 2016

Disclaimer

Harmonic reserves the right to alter the product 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 does 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 product in the manner described herein will not infringe any patents of third parties.

Third-Party Product Trademarks

Adobe® After Effects®, Photoshop®, Flash® Professional, Premiere®

Avid® Media Composer®

Jünger Audio™

Apple® QuickTime®

Microsoft® Mediaroom®

Microsoft® PlayReady®

DOCSIS® 3.0

Start Over® TV

Dolby is a registered trademark of Dolby Laboratories. Dolby Digital, Dolby Digital Plus, Dolby Plus, aacPlus, AC-3, and Dolby® E are trademarks of Dolby Laboratories.

Level Magic and Jünger are trademarks of Jünger Audio Studiotechnik GmbH.

MPEG Audio technology licensed from Fraunhofer IIS http://www.iis.fraunhofer.de/amm/

PitchBlue® is a registered trademark of Vigor Systems.

QuickTime and the QuickTime logo are trademarks or registered trademarks of Apple Computer, Inc., used under license therefrom.

Third-Party Copyright Notes

Harmonic software uses version 3.15.4 of the FreeImage open source image library under FreeImage Public License (FIPL). See http://freeimage.sourceforge.net for details.

The product may include implementations of AAC and HE-AAC by Fraunhofer IIS; and MPEG Audio technology licensed from Fraunhofer IIS

The software described in this publication may use version 2.8 of FFmpeg open source package under Lesser General Public License (LGPL).

The software described in this publication is furnished under a nondisclosure agreement, or the License Agreement and Limited Warranty stated below, and the end user license agreement (which is furnished with the software), which may have additional terms. The software may be used or copied only in accordance with the terms of those agreements. By using the software, you acknowledge you have read the end user license agreement and the License Agreement and Limited Warranty provision.

The product described in this publication may be covered by one or more of U.S. Patents, their foreign counterparts and pending patent applications.

The product 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 product or otherwise fulfilling their conditions constitutes your acceptance of it, 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 section of the documentation directory of the product, user guide, or by contacting us at support@harmonicinc.com.

Notice

Information contained in this publication 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

This document consists of the following sections:

- *Introduction* outlines the guide, lists terms and conventions, provides customer service information, and lists the new commands in this version of the Spectrum Player API.
- *Clip Functions* provides a comprehensive list, descriptions, and examples of Player Control API Clip functions.
- *Player Motion Functions* provides Spectrum API Functions related to the manipulation of Players.
- *Player Status Functions* provides Spectrum API functions that request the status of the Players.
- *Player Timeline Functions* provides a comprehensive list, descriptions, and examples of Player Control API Timeline functions.
- *Player Discovery and Other Functions* provides API Functions related to Player Discovery, and also includes functions that are not tied to a particular Player.
- *OPC Tester* describes the Polaris Play: MediaFetch (MediaFetch) Tester, a simple command line tool that is used for exploring the function set.

Spectrum Player API

This function set is the **Spectrum Player Application Programming Interface** (also known as the **Polaris Play: MediaFetch (MediaFetch) API**). This section introduces the Spectrum concepts of **Player**, **Timeline** and **Clip**. Other sections provide a more detailed discussion of these important concepts. The bulk of the document is comprised of pages that detail each function within the set. The complete **Software Development Kit** (**SDK**) includes the API documentation, the **MediaFetch Tester** application and the necessary source files, header files, library files, and DLL executable. The MediaFetch Tester application is a simple command line tool that is used for exploring the function set; it also provides sample code.

What is a Player?

A Player is a software object inside the Spectrum video server. It is responsible for pushing the media out onto the network that connects the video server to an I/O module. The Player broadcasts the media out onto the bus; assigned I/O modules are configured to listen for the broadcast. Thus, when you want to see a picture coming from an I/O module, you execute functions on a Player object inside the video server to accomplish this.

What is a Clip?

Spectrum uses the term **Clip** to describe a piece of media ("essence" is another term for media). The term clip is used in two different contexts:

- There is **Clip** as a permanent object that has been stored in the Spectrum file system.
- There is **Clip** as a temporary object that has been attached to the **Timeline** that is associated with an Spectrum Player.

The functions that deal with the Clip as a file object all start with the prefix **OmPlrClip** whereas the other functions use the shorter prefix **OmPlr**. The function descriptions for the **OmPlrClip** functions are listed in their own section.

What is a Timeline?

The Spectrum **Timeline** is a software object that is used to play out lists of Clips. Each Player object is created with its own Timeline object. Most of the Player's personality is wrapped up in its Timeline. For instance, a Player doesn't play clips — instead it "plays" its Timeline and clips are attached to the Timeline.

What is the Player API?

The Spectrum Player API is an API rather than a protocol. Instead of sending messages, you call functions in order to control the Player. You need Spectrum software to use the API; this is furnished with the SDK. Your software will be compiled with a set of Spectrum-furnished header files that define constants and that give function prototypes. Your software will be linked with a Player API library (**omplrlib.lib** on Windows, **libomplrlib.a** on Linux). The Windows version of the Player API requires the associated **omplrlib.dll** to be in the path when the application runs.

Required Environment for Using the API

The DLL executable that is supplied with the SDK is used when you run your application. The DLL supports the function calls that your application makes to control the Spectrum Player, such as **OmPlrClipExists**. When you compile the sources of your application, you will use the "include" files that are part of the SDK. These include files provide the function prototypes and define argument values. When you link the objects of your application, you will link with a library file that is provided with the SDK; this library file provides the information that your application will need so that it can later create the dynamic links to the **omplrlib.dll**.

To co-exist with the Harmonic SDK files, you must develop applications with one of the following:

- Microsoft Visual C++ 6.0 (32-bit only)
- Microsoft Visual Studio 10 (32- and 64-bit)
- Gcc-hosted on Linux platforms (32- and 64-bit)

NOTE: Other compilers and languages are likely to work, however, the developer must write the interface code necessary to load the DLL and invoke its functions.

Also note the following:

- Your code and your compiler must be able to co-exist with C++ code. The Spectrum supplied header files include classes; they also use such things as the C++ approach of automatically assigning "typedefs" for each structure and enumeration.
- The Spectrum SDK assumes that the size of "int" is 32 bits. Enumerated values and pointers 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.

Spectrum System Software Compatibility

The new or revised API functionality requires Spectrum system software release 6.1 or later.

Files Included in the SDK Package

The SDK package includes the necessary header, library, and dll files. It also includes the source code for the **OpcTester** application; this source code can be used as example code. Included with the **OpcTester** files are the matching Microsoft VC++ project files and Linux makefile so that you can easily compile and modify the OpcTester files.

The supplied files are organized into a suggested directory tree structure; this structure was used for the OpcTester project. As of this date, the files included are:

The "include" directory consists of the following files:

```
omdefs.h
omplrclnt.h
omplrdefs.h
omtcdata.h
ommediadefs.h
```

These files are needed for compiling.

The "lib" directory holds:

```
omplrlib.lib (Window) or libomplrlib.a (Linux)
```

This file is needed for linking.

The "bin" directory holds:

```
omplrlib.dll (Windows only)
OpcTester.exe (Windows only)
```

The DLL file is needed for running your application. The .exe file is the executable for the OpcTester application. Note that for Linux, the OpcTester application needs to be built from the source.

The "src" directory holds the files for the "opcTester" application, consisting of:

```
3 .cpp files
1 .h file
3 files related to the m/soft VC++ project (Windows only)
1 makefile to build OpcTester (Linux only)
```

These files contain sample code.

ONC RPC Information

The omplrlib.dll that is distributed with the SDK includes the ONC RPC package.

FAQs from the Source Package

Q: What is the difference between this ONC RPC and Microsoft RPC?

A: ONC RPC (formerly called Sun RPC) is a de-facto RPC standard for Unix. It is included in all major Unix distributions. Microsoft RPC (MS RPC) is a different RPC system and it is interoperable with OSF/DCE RPC. OSF/DCE is also available for nearly all operating systems, but usually it is not part of the standard distribution. OSF/DCE RPC is newer and a little bit more powerful than ONC RPC, but for most applications both RPC systems will do the job. ONC RPC and OSF/DCE RPC (and thus MS RPC) are *not* inter-operable at all.

Q: What about the copyright? May I use this code in my (commercial) application?

A: You may use, copy or modify ONC RPC for Windows NT according to SUN copyright (shown below). This may include the free use in your (commercial) applications.

Sun RPC License

Please note the following details regarding the Sun RPC License:

Sun RPC is a product of Sun Microsystems, Inc. and is provided for unrestricted use provided that this legend is included on all tape media and as a part of the software program in whole or part. Users may copy or modify Sun RPC without charge, but are not authorized to license or distribute it to anyone else except as part of a product or program developed by the user.

SUN RPC IS PROVIDED AS IS WITH NO WARRANTIES OF ANY KIND INCLUDING THE WARRANTIES OF DESIGN, MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE, OR ARISING FROM A COURSE OF DEALING, USAGE OR TRADE PRACTICE.

Sun RPC is provided with no support and without any obligation on the part of Sun Microsystems, Inc. to assist in its use, correction, modification or enhancement.

SUN MICROSYSTEMS, INC. SHALL HAVE NO LIABILITY WITH RESPECT TO THE INFRINGEMENT OF COPYRIGHTS, TRADE SECRETS OR ANY PATENTS BY SUN RPC OR ANY PART THEREOF.

In no event will Sun Microsystems, Inc. be liable for any lost revenue or profits or other special, indirect and consequential damages, even if Sun has been advised of the possibility of such damages.

Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, California 94043.

Spectrum System Documentation Suite

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

Table 0–1: Spectrum System Documentation Suite

This document...	Provides this information...
Spectrum System Installation Guide	■ System installation ■ Software installation and upgrade details ■ Orientation to system components ■ Troubleshooting system components ■ Specifications for system components
Spectrum Component Replacement Guides	Component replacement instructions for Spectrum devices
Spectrum Quick Reference Guides and the *Spectrum MediaDeck Installation Guide*	■ Front and back panel views of Spectrum devices ■ LED assignments and legends ■ Quick start steps
Spectrum X and ChannelPort Tools User Guide	■ Using FXTool and PreviewTool

Polaris Play: Playlist User Guide	■ Polaris Play: Playlist Control Overview ■ Using Polaris Play: Scheduler ■ Using Polaris Play: Playlist
Spectrum X and ChannelPort Template Authoring Guide	■ Spectrum X and ChannelPort template authoring
Spectrum Release Notes	Last minute information regarding a product release
Spectrum Media and Wrapper Formats	Supported clip types and wrapper formats
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 System Protocol Reference Guide	■ Command sets and preroll parameters for controlling Spectrum servers ■ The Harmonic implementation of FTP server

Software updates and documentation are available from the Harmonic website. Contact Harmonic Technical Support for login information.

Acrobat ® Reader® is needed to view the product documentation. Download this for free from: **http://www.adobe.com**

Technical Support

For information on contacting Harmonic Technical Support, refer to *Appendix B, 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:\Spectrum\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 Spectrum 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 Spectrum 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 Spectrum 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
Clip Functions

This section provides Spectrum API functions relating to the manipulation of clips. Functions described in this section include the following:

Function	Description
OmPlrClipCopy	Copy part or all of a clip into a new clip of a different name.
OmPlrClipCopyAbort	Abort a clip copy operation.
OmPlrClipCopyFree	Obtain a list of all open OmPlrClipCopy Handles.
OmPlrClipCopyEnumerate	Free the handle of a clip copy operation.
OmPlrClipCopyGetParams	Get the parameters of a current clip copy operation.
OmPlrClipCopyGetStatus	Get the current status of a current clip copy operation.
OmPlrClipDelete	Delete clip from storage.
OmPlrClipExists	Queries whether clip exists.
OmPlrClipExtractData	Extract data from a clip, for example closed caption data.
OmPlrClipGetDirectory	Get current clip directory name.
OmPlrClipGetExtList	Get clip extension list for this network connection.
OmPlrClipGetFirst	Returns name of first in “current directory.”
OmPlrClipGetFsSpace	Get free space and total space of the file system.
OmPlrClipGetInfo	Get information about a clip.
OmPlrClipGetInfo1	Get information about a clip including file type and modification time.
OmPlrClipGetMediaName	Get a full filename for a clip or a track within the clip.
OmPlrClipGetNext	Get the names of the remaining clip in the “current” directory.
OmPlrClipGetStartTimecode	Get the timecode of the first frame in a clip.
OmPlrClipSetStartTimecode	Set the start timecode saved in the clip metadata.
OmPlrClipGetTrackUserData	Get track clip user data.
OmPlrClipGetTrackUserDataAndKey	Get track clip user data by index.
OmPlrClipGetUserData	Get user data from a clip that has been stored using a key.
OmPlrClipGetUserDataAndKey	Get user key and data from a clip using a key index.
OmPlrClipInsertData	Insert data into a clip, for example closed caption data.
OmPlrClipRegisterCallbacks	Register function set to call whenever a clip is added or deleted from storage medium.
OmPlrClipRegisterWriteCallbacks	Register callbacks for clip open/close for write changes.

Function	Description
OmPlrClipRename	Used to change the name of or move an existing clip on the storage medium.
OmPlrClipSetDefaultInOut	Set the default in/out points for a clip.
OmPlrClipSetDirectory	Set current clip directory name.
OmPlrClipSetExtList	Set clip extension list for this network connection.
OmPlrClipSetProtection	Set or clear clip protection.
OmPlrClipSetTrackUserData	Set track clip user data.
OmPlrClipSetUserData	Store an arbitrary byte string under a key with a clip.
OmPlrClipWhereRecording	Get the serial number of the server recoding the specified clip.

Clip Function Discussion

Spectrum uses the term **Clip** to describe a piece of media ("essence" is another term for media). The term clip is used in two different contexts:

- There is **Clip** as a permanent object that has been stored in the Spectrum file system, and
- There is **Clip** as a temporary object that has been attached to the **Timeline** that is associated with an Spectrum Player.

Every clip that has been attached to the timeline also exists as a file in the Spectrum file system; this is true even for a newly created file that does not have any media recorded yet. The clip in the Spectrum file system consists of a set of files; there is a "wrapper" file that defines the properties of the clip and then there is a set of media files.

If a clip is known as "ExampleClip", then the wrapper file will be named "ExampleClip.mov." The wrapper file is based on the QuickTime™ standards. You can think of the wrapper file as a database element; it contains information about the clip such as the length of the clip. The wrapper file also contains references for the media files. The media files are kept in a subdirectory of the directory that holds the **.mov** file; the subdirectory is named **media.dir**. The clip may have more than one media type associated with it; the different media types are kept in separate files. For instance, DV25 media will be kept in a file that has an extension of .**dv**; in the case of the dv format, the file can include both video and audio data.

The file names follow Windows 32™ rules. The clip name is unique in its own directory but can be duplicated in another directory. File names can be quite long and most characters are valid:

- It cannot contain the characters / \ : * ? < > |
- It cannot contain the double quote character (")
- All other characters are allowed, including spaces.

The Spectrum Player object treats the names as case sensitive.

Some of the properties of a clip held in the wrapper file include

- The number called **firstFrame** is assigned to the first frame stored in the media files.
- A number called **lastFrame** is assigned to the last frame stored in the media files — plus one.
- The frame rate of the recording.
- A number called the **defaultIn** value. A number called the **defaultOut** value.

- User provided data (if any) is stored in the wrapper file.

The **defaultIn** and **defaultOut** numbers can be used when the clip is prepared for playout by a Player object. Part of the preparation step is to tell the Player which frame to show first and which to show last (or rather, the lowest and highest numbered frames, since the direction of playout could be reverse). The default numbers stored in the wrapper file can be used to provide the required IN and OUT numbers while prepping the file.

There is a function that allows you to retrieve information that is stored in the wrapper file. There is also a function that lets you change the **defaultIn** and **defaultOut** values. There is a separate set of functions for setting and retrieving user data; the user data consists of arbitrary data strings. The other stored clip values cannot be changed; these other numbers are established at the time that the clip was created.

Many **OmPlrClip** functions state "This function can be used even if a NULL Player name was used to open the connection" in the Remarks section for each function. Be aware that you may need to explicitly set the clip directory and the extension list if you use a NULL player and are not happy with the defaults of clip.dir and ".mov". You have to set the directory and extension list for these functions:

- *OmPlrClipGetFirst*
- *OmPlrClipGetNext*
- *OmPlrClipRegisterCallbacks*
- *OmPlrClipRegisterWriteCallbacks* (if the defaults are not satisfactory)

Use the functions *OmPlrClipSetDirectory* and *OmPlrClipSetExtList* to establish non-default values. For other **OmPlrClip** functions that take a clip name argument, you can use fully qualified path names instead of setting directory and extension list when the defaults are not satisfactory.

Clip Name as Function Argument

A number of the API functions have arguments that are a clip name; **OmPlrAttach** and **OmPlrClipGetInfo** are examples. The following rules apply:

- The maximum length is **omPlrMaxClipDirLen** which has a value of 512 characters in length, including the '\0' terminator.
- The argument is case sensitive. Refer to *Clip Function Discussion* for a discussion on characters not allowed.
- The clip name component of the clipName argument should not exceed **omPlrMaxClipNameLen** which has a value of 64 bytes.
- The argument can either be just the clip name (e.g. "fred") or can be the combination of a path plus a clip name plus a valid wrapper extension. For example: /FS1/clip.dir/fred.mov.
- If the argument is just the clip name, then the clip will be assumed to be in the "current clip directory" (see *OmPlrClipSetDirectory*). The current clip directory is associated with **playerHandle**. Each connection to a player has its own handle and has its own current clip directory. A default current clip directory is established by the OmPlrOpen function. In the case of **OmPlrAttach**, when it creates a new clip, the clip will be created in "current clip directory". When the argument is just the clip name, it should not have any extension.
- If the argument is just the clip name, the system selects the first clip found with an extension that matches one from the clip extension list. For all functions, except **OmPlrAttach** and **OmPlrAttach1**, if a match is not found an error is generated. For **OmPlrAttach** and **OmPlrAttach1**, if a match is not found a clip is created using the first extension from the current clip extension list. The current clip extension list is associated

with **playerHandle**. Each connection to a player has its own handle and has its own current clip extension list. A default clip extension list is established by the **OmPlrOpen** function.

- If the argument starts with a "/", then it is assumed to be the absolute path to the clip followed by the clip name. The path part of the argument follows the rules discussed in **OmPlrClipSetDirectory**.

OmPlrClipCopy

Description

Copies part or all of a clip into a new clip of a different name.

```
OmPlrError OmPlrClipCopy(
OmPlrHandle playerHandle,
const TCHAR *pSrcClipName,
const TCHAR *pDstClipName,
uint srcStartFrame,
uint copyLength,
uint dstStartFrame,
bool dummy,
OmPlrClipCopyHandle * pCcHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pSrcClipName	A pointer to a null-terminated string that gives the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy". The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to **512**.
pDstClipName	A pointer to a null-terminated string that gives the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip copy operation will be into the same directory as the source clip. The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512. The media.dir subdirectory will be created if it does not already exist; the command will return failure if any of the other directories do not already exist.
srcStartFrame	A numeric value referencing the first frame of the source clip to be copied. The special value ~0 (0xFFFFFFFF) means start at the first frame in the clip.

Parameter	Description
copyLength	A numeric value giving the number of frames to be copied from the source clip. The special value ~0 (0xFFFFFFFF) means a length of start to the last frame in the clip; this value should be used if making a copy of a clip that is still being recorded. A value of 0 will create a zero length destination clip. A value that exceeds the duration of the source clip will cause the command to fail with an error of **omPlrClipCopyRangeError**.
dstStartFrame	A numeric value giving the number that will be used to refer to the first frame of the destination clip. Use the special value of ~0 (0xFFFFFFFF) to have the dstStartFrame value be the same as the StartFrame number for the source clip.
dummy	A bool value that does not matter. This argument is obsolete but retained for historic purposes.
pCcHandle	An optional pointer that is used to return a numeric value. An arbitrary numeric value is assigned by the player for each clip copy operation. This number is returned to the caller to be used as a handle; the handle will be needed as a parameter for future calls to check on the status of the clip copy operation. **NOTE:** A valid handle will never have a value of 00. Nothing will be written to the pointer if the function returns an error.

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file playerdefs.h. The error codes are assigned in ranges. The range for player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This function can be used to copy within one clip directory or from one clip directory to another. It cannot copy from another MediaDirector. If the destination clip is created in the same directory as the source clip, then the **dstClipName** must be different from the **srcClipName**.

The copy is an asynchronous deep copy (i.e., the media is copied); the copy proceeds at faster than real time but is not instantaneous. The rate of copying will depend on how much other data is being accessed off the media storage; the clip copy is handled at a lower priority than providing media for real time playout. Up to four copies can be in process at the same time on a MediaDirector; **OmPlrClipCopy** will return an error if you try to start a fifth copy.

The copy handle value that is returned via the pointer can be used to obtain copy completion status. The handle should be freed when it is no longer needed.

Use **srcStartFrame** = ~0, **copyLength** = ~0, and **dstStartFrame** = ~0 to make an exact copy. Use **srcStartFrame** = ~0, **copyLength** = ~0, and **dstStartFrame** = 0 to copy entire clip to a new destination clip that starts with a **firstFrame** of 0.

A source clip that is still being recorded can be copied while it is recording. The copy process follows behind the record point and not exceed it. The copy terminates when the source clip stops growing. A copy of a clip cannot be started if the source clip is in the **CueRecord** state.

This function and the other **OmPlrClipCopy** functions can be used even if a NULL Player name was used to open the connection.

CAUTION: OmPlrClipCopy does not support clip properties, such as "low latency." For workflows in which this type of clip is required, please use the Media API version 7.2 or greater.

NOTE: OmPlrClipCopy preserves the extension of the source clip regardless of the extension list setting. For example, if the source clip argument of "fred" matches to a clip "fred.mov", and the destination clip is "sally", then the OmPlrClipCopy operation will create a clip named "sally.mov". The extension of the source clip is also preserved when the source argument is a fully qualified clip such as "/FS1/clip.dir/fred.mov". Note that specifying a mismatched set of extensions such as a source of "FS1/clip.dir/fred.mov" and a destination of "/FS1/clip.dir/sally.mxf" is not supported

CAUTION: This function will not return an error if some of the media files are missing. However, the copy status will indicate fewer bytes copied than expected.

NOTE: If OmPlrClipCopy is used to copy a clip being recorded on a MediaDirector from another MediaDirector connected to the same file system: 1) configure all of the MediaDirectors to use an existing NTP servier if one is available, and 2) synchronize the clocks on the two MediaDirectors. Refer to the *Spectrum SystemManager Installation Guide* for step-by-step instructions on how to install NTP for Windows to ensure synchronicity.

See also

OmPlrClipSetDirectory, *OmPlrClipCopyAbort*, *OmPlrClipCopyFree*, *OmPlrClipCopyGetStatus*

OmPlrClipCopyAbort

Description

Aborts a clip copy operation.

```
OmPlrError OmPlrClipCopyAbort(
OmPlrHandle playerHandle,
OmPlrClipCopyHandle ccHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
ccHandle	The clip copy handle that was returned by the call to *OmPlrClipCopy*.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file playerdefs.h. The error codes are assigned in ranges. The range for player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This command will abort an existing clip copy operation. The destination clip will NOT be deleted; if the operation had not completed, the destination clip will be incomplete and will not play back correctly. In most cases, you should delete the destination clip after an abort. Harmless to call if the clip copy operation has already completed. Does not free the clip copy handle; you still need to call *OmPlrClipCopyFree* to release the clip copy handle.

See also

OmPlrClipCopy, *OmPlrClipCopyGetStatus*, *OmPlrClipCopyFree*

OmPlrClipCopyFree

Description

Frees the handle of a clip copy operation.

```
OmPlrError OmPlrClipCopyFree(
OmPlrHandle playerHandle,
OmPlrClipCopyHandle ccHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
ccHandle	The clip copy handle that was returned by the call to *OmPlrClipCopy*.

Return value

A numeric value that defines success or error. A value of 0 indicates “all ok.” Error codes are defined in the file playerdefs.h. The error codes are assigned in ranges. The range for player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

If the clip copy is still in progress, first use **OmPlrClipCopyAbort**; otherwise **OmPlrClipCopyFree** will return an error.

See also

OmPlrOpen, *OmPlrClipCopy*, *OmPlrClipCopyAbort*, *OmPlrClipCopyGetStatus*

OmPlrClipCopyEnumerate

Description

Obtains a list of all open **OmPlrClipCopyHandles**.

```
OmPlrError OmPlrClipCopyEnumerate(
OmPlrHandle playerHandle,
OmPlrClipCopyHandle ccHandle[],
uint numHandles,
uint *pNumRetHandles);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
ccHandle[]	An array that can hold **OmPlrClipCopyHandle** values. Pass a pointer to this array.

Parameter	Description
numHandles	The size of the array, in terms of how handles it can hold.
pNumRetHandles	An required pointer used to return the number of handles that were written into the array. No value is returned if the function returns an error.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file playerdefs.h. The error codes are assigned in ranges. The range for player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

If the MediaDirector returned more clip copy handles than the value of **numHandles**, then an error of **omPlrBufferTooSmall** will be returned.

See also

OmPlrClipCopy, *OmPlrClipCopyGetStatus*, *OmPlrClipCopyFree*

OmPlrClipCopyGetParams

Description

Gets the parameters for a current clip copy operation.

```
OmPlrError OmPlrClipCopyGetParams(
OmPlrHandle playerHandle,
OmPlrClipCopyHandle ccHandle,
TCHAR *pSrcClipName,
uint srcClipNameSize,
TCHA *pDstClipName,
uint dstClipNameSize,
uint *pSrcStartFrame,
uint *pCopyLength,
uint *pDstStartFrame,
bool *pDummy);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
ccHandle	The clip copy handle that was returned by the call to *OmPlrClipCopy*.
pSrcClipName	An optional pointer to a string buffer that will receive the name of the source clip.
srcClipNameSize	The size of the **srcClipName** buffer, in units of characters. The command returns an error if a pointer was passed and the buffer was too small. This parameter is ignored if the **pSrcClipName** pointer is passed as NULL.

Parameter	Description
pDstClipName	An optional pointer to a string buffer that will receive the name of the destination clip.
dstClipNameSize	The size of the **dstClipName** buffer, in units of characters. The command returns an error if a pointer was passed and the buffer was too small. This parameter is ignored if the **pDstClipName** pointer is passed as NULL.
pSrcStartFrame	An optional pointer used to return a unsigned 32 bit integer value. The value identifies the first frame of the source clip that was copied.
pCopyLength	An optional pointer used to return a unsigned 32 bit integer value. The value is the number of frames that will be copied when this operation is complete.
pDstStartFrame	An optional pointer used to return a unsigned 32 bit integer value. The value was the **dstStartFrame** parameter for the **copyClip** command and it is the number that identifies the first frame of the destination clip.
pDummy	An optional pointer used to return a Boolean value. This argument is obsolete but retained for historic purposes.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file playerdefs.h. The error codes are assigned in ranges. The range for player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

No values are written back using the pointers if the function returns an error.

If a "~0" argument was used for the **copyLength** parameter of the clipCopy command, then this function will return a "~0" value. The same thing also happens for the **srcStartFrame** and **dstStartFrame** arguments.

See also

OmPlrOpen, *OmPlrClipCopy*, *OmPlrClipCopyGetStatus*

OmPlrClipCopyGetStatus

Description

Gets the current status of a clip copy operation.

```
OmPlrError OmPlrClipCopyGetStatus(
OmPlrHandle playerHandle,
OmPlrClipCopyHandle ccHandle,
uint *pNumFramesCopied,
uint *pCopyLength,
bool *pDone,
OmPlrError *pDoneStatus);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
ccHandle	The clip copy handle that was returned by the call to *OmPlrClipCopy*.
pNumFramesCopied	An optional pointer used to return the number of frames copied so far.
pCopyLength	An optional pointer used to return the value of the **copyLength** parameter that was used in the **OmPlrCopyClip** call.
pDone	An optional pointer used to return a value that is true when the operation is complete.
pDoneStatus	An optional pointer used to return a error code when the operation is complete.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file playerdefs.h. The error codes are assigned in ranges. The range for player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

No value is returned using the pointer parameters if the function returns an error. If a "~0" argument was used for the **copyLength** parameter of the **clipCopy** command, this function returns a "0" value for **copyLength**. The value of **CopyLength** can be used with **NumFrameCopied** to form a fraction representing percentage completed. The **DoneStatus** is only valid after the *Done* value indicates completion.

See also

OmPlrOpen, *OmPlrClipCopy*

OmPlrClipDelete

Description

Used to delete a clip from the storage medium.

```
OmPlrError OmPlrClipDelete(
OmPlrHandle playerHandle,
const TCHAR* pClipName);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipname	A pointer to a null-terminated string that gives the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

The **OmPlrDelete** function will fail, indicating **omPlrFailure**, if the clip had its protection property enabled.

The **OmPlrDelete** function will indicate success if asked to delete a clip that does not exist.

Remarks

This function *permanently* removes the clip from the storage. It removes all components of the clip, such as the ".mov" wrapper file and all the media files. Use **OmPlrDetach** if your goal is to just remove the clip from the timeline of a Player.

NOTE: This function can be used even if a NULL Player name was used to open the connection.

It is not illegal to delete a clip that is in use. If you call **OmPlrClipDelete** for a clip that is currently attached to a timeline, some special rules come into effect. In all cases, the clip will immediately become unavailable for new loads by **OmPlrAttach** and a query made with **OmPlrClipExists** will return "does not exist." If the clip was the "on-air" clip at the time that it was deleted, then the media data will not be deleted until after the clip has gone "off-air." On the other hand, if the clip was attached but was not yet "on-air" (i.e., not yet viewable), then the media data will be immediately deleted and black/silence will be used for that clip if it later goes on-air. If a clip was being recorded when it was deleted, the recording will proceed but will not generate any media files. A deleted clip will always be reported by **OmPlrGetClipName** and **OmPlrGetPlayerStatus** as having an empty name.

See also

OmPlrOpen, *OmPlrClipSetDirectory*

OmPlrClipExists

Description

Queries whether a clip exists on the storage medium.

```
OmPlrError OmPlrClipExists(
OmPlrHandle playerHandle,
const TCHAR *pClipName,
```

```
bool *pExists);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipname	A pointer to a null-terminated string that gives the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
pExists	A pointer used to return a boolean value. A value of true will be returned if the clip exists in the current directory; otherwise a value of false will be returned. No value is written if an error is encountered.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok". Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

A newly created clip that is still empty will already exist as a file; that file will be automatically deleted if the clip is detached while still empty.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrOpen, *OmPlrClipSetDirectory*

OmPlrClipExtractData

Description

Extracts data such as closed caption data from a clip.

```
OmPlrError OmPlrClipExtractData(
OmPlrHandle lrHandle,
const TCHAR *pClipName,
uint startFrame,
uint numFrames,
OmPlrClipDataType dataType,
unsigned char *pData,
uint *pDataSize);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipname	A pointer to a null-terminated string that gives the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
startFrame	A numeric value that provides the frame number to start data extraction from the clip.
numFrames	A numeric value that determines the number of frames to extract from a clip. This value can range from 1 to 512.
dataType	Determines the type of data to extract as specified by enum **OmPlrClipDataType** in the omplrdefs.h file.
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. The amount of returned data is number of frames (**numFrames**) times the number of bytes per frame. For example, extracting 100 frames of CC data (**omPlrClipDataCcFrame**) is 100 x 4 = 400 bytes.
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

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file playerdefs.h. The error codes are assigned in ranges. The range for player errors starts at PLAYER_ERROR_BASE (0x00009000).

Returns the amount of data extracted in bytes.

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 **OmPlrClipExtractData**.

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 **OmPlrClipExtractData** each extracting 200 frames worth of CC starting at frame locations 0, 200, 400, 600... 9800.

OmPlrClipExtractData is declared in **omplrclnt.h**.

See also

OmPlrClipInsertData, *OmPlrOpen*, *Clip Name as Function Argument*

OmPlrClipGetDirectory

Description

Gets the current clip directory name.

```
OmPlrError OmPlrClipGetDirectory(
OmPlrHandle playerHandle,
TCHAR *pClipDir,
uint clipDirSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipDir	A pointer to a null-terminated string that gives the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. In the case of any error, nothing will be written to the buffer.
clipDirSize	Gives the size of the buffer that **pClipDir** points at. If the buffer is smaller than the length of the clip directory string (inclusive of the terminating '\0'), then the function will fail and will return an error of **omPlrBufferTooSmall**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The returned path name will be an ABSOLUTE path name. Each instance of **playerHandle** has its own "current clip directory." The default value established on connection to the Player is "/xxx/clip.dir" where "xxx" is the primary file system name. The constant **omPlrMaxClipDirLen** sets the limit for the longest directory name and is defined in the file omplrdefs.h; its current value is 512, which includes the terminating '\0' character.

This function can be used even if a NULL Player name was used to open the connection.

See also

Clip Function Discussion, *OmPlrClipSetDirectory*

OmPlrClipGetExtList

Description

Gets the clip extension list for a MediaDirector.

```
OmPlrError OmPlrClipGetExtList(
OmPlrHandle plrHandle,
```

```
            TCHAR *pExtList,
            uint extListSize);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pExtList	A pointer to a buffer that is used to store the returned extension list.
extListSize	A numeric value that shows the size in bytes of the **pExtList** buffer.

Return value

Returns a list of filename extensions that can be used with the system.

Remarks

The extension list is a concatenated list of filename extensions that are used as clip identifiers. For example: for QuickTime files and .dv files the list would look like ".mov.dv". This clip extension list is an attribute of the network connection to the MediaDirector, not an attribute of the player. Clip names specified in other functions use this list of extensions. This function sets the **extListSize** to the size (in bytes) of the returned extension list.

See also

OmPlrOpen

OmPlrClipGetFirst

Description

Returns the name of the first clip on the storage medium in "current directory."

```
OmPlrError OmPlrClipGetFirst(
OmPlrHandle playerHandle,
TCHAR *pClipName,
uint clipNameSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipName	A pointer to a buffer that gives the name of the first clip. For non Unicode builds, this is an 8-character string. For Unicode builds, this is a 16-bit character string. In the case of error, nothing will be written to the buffer.
clipNameSize	Gives the size of the buffer that **pClipName** points at. If the buffer is smaller than the length of the clip name string (inclusive of the terminating '\0'), then the function will fail and will return an error of **omPlrBufferTooSmall**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." A value of **omPlrEndOfList** indicates that there are no clips available. Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Returns the name of the first clip that has any of the extensions found in the current extension list. For instance if the extension list is ".mxf.mov" and the first clip encountered was "fred.mov" then the function **OmPlrClipGetFirst** would return "fred". Note that this function does not return the extension.

Remarks

Clips are enumerated by first calling **OmPlrClipGetFirst** and then repeatedly calling **OmPlrClipGetNext**. Keep calling until the return value is **omPlrEndOfList**. The call that returns the value **omPlrEndOfList** will not return any name in the buffer.

The clip names are returned in the order of the creation date.

Only the clip names in the "current clip directory" are returned. The "current clip directory" is associated with **playerHandle**.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrClipGetNext, *OmPlrClipSetDirectory*

OmPlrClipGetFsSpace

Description

Gets the free space and total space of the file system.

```
OmPlrError OmPlrClipGetFsSpace(
OmPlrHandle playerHandle,
uint_64* pTotalBytes,
uint_64* pFreeBytes);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pTotalBytes	An optional pointer used to return a 64bit binary number that gives the total size of the file system in units of bytes. Use a 0 value if you are not interested in this number.
pFreeBytes	An optional pointer used to return a 64-bit binary number that gives the amount of free space on the file system in units of bytes. Use a 0 value if you are not interested in this number.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The file system space is returned in units of BYTES. To convert free space from bytes to time requires that an estimate be made based on file formats and video compression rates. The function **OmPlrGetRecordTime** makes such an estimate.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrOpen, *Clip Function Discussion*, *Player Function Discussion*, *OmPlrGetRecordTime*

OmPlrClipGetInfo

Description

Used to get information about a clip on the storage medium.

```
OmPlrError OmPlrClipGetInfo(
OmPlrHandle playerHandle,
const TCHAR *pClipName,
OmPlrClipInfo *pClipInfo);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipname	A pointer to a null-terminated string that gives the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
pClipInfo	A pointer to a buffer that holds a OmPlrClipInfo structure. The function will overwrite the buffer contents with the returned values of the OmPlrClipInfo structure. **IMPORTANT**: Some of the fields in the buffer need to be initialized before you call the OmPlrGetClipInfo function because they determine how much information is returned.

Parameter	Description
pClipInfo ->size	This field within the **OmPlrClipInfo** structure must be initialized with the size of the structure before calling the **OmPlrClipGetInfo** function.
pClipInfo->ms **pClipInfo->maxMsTracks**	**pClipInfo->ms** is a pointer to a buffer that can hold a number of **OmMediaSummary** structures. An **OmMediaSummary** structure provides information about the format of the media tracks. If you need this information, you must: 1 Pick a limit for the maximum number of media tracks that you want to handle. The server code will never return more than 32 tracks of information (as of this date). 2 Allocate a buffer large enough for this number of **OmMediaSummary** structures. 3 Store a pointer to this buffer into the ms member of **OmPlrClipInfo** structure that you have allocated before you call the function **OmPlrClipGetInfo**. 4 Place the chosen number into the maxMsTracks member of the **OmPlrClipInfo** structure that you have allocated before you call the function **OmPlrClipGetInfo**. If you do not want any information about the media tracks, then you must: ■ Place a 0 value into the maxMsTracks member of **OmPlrClipInfo** structure that you have allocated before you call the function **OmPlrClipGetInfo**. ■ The **ms** member of the **OmPlrClipInfo** structure that you have allocated can be left with a garbage value. Other fields in the structure do not need to be initialized. The fields will all be overwritten if the function returns successfully.

A numeric value that defines success or error. A value of 0 indicates “all ok.” Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

On a successful return the **pClipInfo->maxMsTracks** field will have changed and now shows the number of tracks in the clip.

This function can be used even if a NULL Player name was used to open the connection.

The structure **OmPlrClipInfo** itself and the structure and enums used in **OmPlrClipInfo** are currently defined in omplrdefs.h. Following are descriptions and additional information for each field in the structure.

size	Gives the size of the structure. This *must* be initialized.
version	For internal use only.
firstFrame	Gives the first recorded frame.
lastFrame	Gives the last recorded frame.

defaultIn	Gives the Default in point (inclusive, first frame to play).
defaultOut	Gives the Default out point (exclusive, last frame to play).
numVideo	Gives the number of video tracks.
numAudio	Gives the number of audio tracks.
frameRate	Gives the clip frame rate.
protection	Shows **True** if protected. You cannot delete the clip if it is protected.
notOpenForWrite	Shows **True** if not open for write.
notReadyToPlay	Shows **True** if not ready to play.
res1	This parameter is reserved.
creationTime	Shows the creation time, in seconds from Jan 1, 1970.
maxMsTracks	Gives the number of elements in the following *ms* array. This *must* be initialized.
***ms**	This points to an array of **OmMediaSummary** structures

OmMediaSummary is a structure that provides detailed information about tracks in a clip. The structure is currently defined in omplrdefs.h. Following are descriptions for each field in the structure:

type	Shows the basic type of media contained in the track.
vsr	This is an enumeration indicating the video sample ratio for example: 4:2:2.
aspect	This is an enumeration indicating the *picture* aspect ratio (not to be confused with the pixel aspect ratio).
bitrate	Gives the bitrate in bits per second for the track. Shows 0 if the audio is embedded.
bitsPerUnit	For audio tracks, shows the sample size in bits (for example, 16, 24, 32 bits). For video tracks, shows the pixel size in bits (for example, 8 or 10 bits). Otherwise, shows 0.
sampleRate	Gives the media sample rate. Shows, for example, 2997 for NTSC video, 4800000 for 48KHz audio.
channels	Gives the number of audio channels; shows 1 for video, N for audio.
specific.mpeg.gopLength	Gives the length of the GOP structure. Typically shows 15 for Long GOP MPEG video, 0 for everything else.
specific. mpeg.subGopLength	Gives the length of the sub-GOP structure. Usually shows 3 for MPEG, 0 for everything else.
specific.audio.format	This is an enumeration indicating the audio type.

specific.audio.bigEndian	Shows 0 when audio samples are stored with the least significant byte first (Little Endian). Shows 1 when audio samples are stored with the most significant byte first (Big Endian).
specific.audio.sampleStride	Gives the distance in bytes between audio samples. Typically this is the same as the sample size, but it could possibly be larger.
specific.vbi.lineMask	This is a 32-bit mask indicating which video lines encode VBI. Use the expression "1 << (line - 1)" to test.

See also

OmPlrOpen, *OmPlrClipGetDirectory*, *OmPlrClipSetDirectory*, *OmPlrClipGetInfo1*

OmPlrClipGetInfo1

Description

Used to get information about a clip on the storage medium including file type and modification time.

```
OmPlrError OmPlrClipGetInfo1(
OmPlrHandle playerHandle,
const TCHAR *pClipName,
OmPlrClipInfo1 *pClipInfo);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipname	A pointer to a null-terminated string that gives the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.

Parameter	Description
pClipInfo	A pointer to a buffer that holds a **OmPlrClipInfo1** structure. The function will overwrite the buffer contents with the returned values of the **OmPlrClipInfo1** structure. **IMPORTANT:** Some of the fields in the buffer need to be initialized before you call the OmPlrGetClipInfo function because they determine how much information is returned.
pClipInfo->ms **pClipInfo->maxMsTracks**	**pClipInfo->ms** is a pointer to a buffer that can hold a number of **OmMediaSummary** structures. An **OmMediaSummary** structure provides information about the format of the media tracks. If you need this information, you must: 1 Pick a limit for the maximum number of media tracks that you want to handle. The server code will never return more than 32 tracks of information (as of this date). 2 Allocate a buffer large enough for this number of **OmMediaSummary** structures. 3 Store a pointer to this buffer into the ms member of **OmPlrClipInfo** structure that you have allocated before you call the function **OmPlrClipGetInfo**. 4 Place the chosen number into the maxMsTracks member of the **OmPlrClipInfo** structure that you have allocated before you call the function **OmPlrClipGetInfo**. If you do not want any information about the media tracks, then you must: ■ Place a 0 value into the maxMsTracks member of **OmPlrClipInfo** structure that you have allocated before you call the function **OmPlrClipGetInfo**. ■ The **ms** member of the **OmPlrClipInfo** structure that you have allocated can be left with a garbage value. Other fields in the structure do not need to be initialized. The fields will all be overwritten if the function returns successfully.

Return value

A numeric value that defines success or error. A value of 0 indicates “all ok.” Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

On a successful return the **pClipInfo->maxMsTracks** field will have changed and will now show the number of tracks in the clip.

This function can be used even if a NULL Player name was used to open the connection.

The structure **OmPlrClipInfo1** itself and the structure and enums used in **OmPlrClipInfo1** are currently defined in omplrdefs.h**. OmPlrClipInfo**1 contains the same fields as **OmPlrClipInfo** and the additional fields described below:

result	Gives the first recorded frame.
fileType	Gives the movie file type. For example, MXF, QuickTime.
mediaContainment	This is an enumeration that indicates if media essence is embedded in the clip file, contained in an external file, or both.
modificationTime	Gives the last modification time, in seconds, from January 1, 1970.
***ms**	Points to an array of OmMediaSummary1 structures.

OmMediaSummary1 is a new version of the **OmMediaSummary** structure providing information about width and height of video frames. The structure is currently defined in omplrdefs.h. **OmMediaSummary1** contains the same fields as **OmMediaSummary** and the additional fields described below

Following are descriptions and additional information for each parameter:.

specific.video. width	Gives the width in pixels of video frames.
specific.video. height	Gives the height in pixels of video frames.

CAUTION: Do not forget to initialize the pClipInfo-> maxMsTracks member in the OmPlrClipInfo1 structure.

See also

OmPlrOpen, *OmPlrClipGetDirectory*, *OmPlrClipSetDirectory*, *OmPlrClipGetInfo*

OmPlrClipGetMediaName

Description

Gets a full filename for a clip or a track within a clip.

```
OmPlrError OmPlrClipGetMediaName(
OmPlrHandle playerHandle,
const TCHAR *pClipName,
uint trackNum,
uint fileNum,
TCHAR *pMediaName,
uint mediaNameSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipname	A pointer to a null-terminated string that gives the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
trackNum	Identifies the track in the clip. Tracks are consecutively numbered starting from 0. Track 0 is for the QuickTime movie file. Tracks 1 to N are for the N video tracks. Tracks N+1 to N+M are for the M audio tracks.
fileNum	Each track typically only has one associated file but can have more than one file associated. A track that has been formed by splicing together different media segments may have more than one file. The **fileNum** argument identifies which file that is associated with the indicated track should be examined by this function. Files are consecutively numbered starting from 0.
pMediaName	A pointer to a buffer that will be used for returning the name of the media file. The name will include the full path. For non Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. In the case of any error, nothing will be written to the buffer. Maximum length is **omPlrMaxClipDirLen** (512).
mediaNameSize	Gives the size of the buffer (in bytes) that **pMediaName** points at. If the buffer is smaller than length of the media file name (including the full path and the terminating '\0'), then the function will fail and will return an error of **omPlrBufferTooSmall**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." A value of **omPlrEndOfList** indicates that either **trackNum** or **fileNum** is past the limits within this clip. Other Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Use *OmPlrClipGetInfo* to find the number of video and audio tracks.

If you want to obtain all the media file names, build a double loop that iterates over the tracks in the outer loop starting from 0 and that iterates over file numbers in the inner loop starting from 0. Break out of the inner loop if the value omPlrEndOfList is returned. Break out of the outer loop if the value omPlrEndOfList is returned for a fileNum of 0.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrClipGetInfo

OmPlrClipGetNext

Description

Gets the names of the “rest of” the clips on the storage medium in the current directory.

```
OmPlrError OmPlrClipGetNext(
OmPlrHandle playerHandle,
TCHAR *pClipName,
uint clipNameSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipName	A pointer to a buffer that will be used for returning the name of the next clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. In the case of any error, nothing will be written to the buffer.
clipNameSize	Gives the size of the buffer that **pClipName** points at. If the buffer is smaller than length of the clip name string (inclusive of the terminating ‘\0’), then the function will fail and will return an error of **omPlrBufferTooSmall**.

Return value

A numeric value that defines success or error. A value of 0 indicates “all ok.” A value of **omPlrEndOfList** indicates that the last of the names was returned on the previous call. Other Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Returns the name of the next clip that has any of the extensions present in the current extension list. For instance, if the extension list is “.mxf.mov” and the next clip encountered is “fred.mov” then the function **OmPlrClipGetNext** would return “fred”. Note that this function does not return the extension. In the case of an extension list such as “mxf.mov” and a clip directory that holds both .mxf and .mov clips, the function **OmPlrClipGetNext** would return the clip name interspersed by type.

Remarks

Clips are enumerated by first calling **OmPlrClipGetFirst** and then repeatedly calling **OmPlrClipGetNext**. Keep calling until the return value is **omPlrEndOfList**. The call that returns the value **omPlrEndOfList** will not return any names in the buffer.

The clip names are returned in the order the creation date.

Only the clip names in the “current clip directory” that was in effect at the time of the **OmPlrClipGetFirst** function call are returned. The “current clip directory” is associated with **playerHandle**. Note that changes to the “current clip directory” after the **OmPlrClipGetFirst** won’t effect the clips returned by the **OmPlrClipGetNext** function.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrOpen, *OmPlrClipGetFirst*, *OmPlrClipSetDirectory*

OmPlrClipGetStartTimecode

Description

Gets the start timecode saved in the clip metadata. The start timecode in the clip metadata is usually the timecode from the first frame of the clip. Note this can be changed using *OmPlrClipSetStartTimecode*.

```
OmPlrError OmPlrClipGetStartTimecode(
OmPlrHandle plrHandle,
const TCHAR q*pClipName,
OmTcData *pStartTc;
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipName	A pointer to a null-terminated string that gives the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see the *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
pStartTC	A pointer that is used to return the first timecode from the given clip. The buffer pointed at needs to be at least as large as the **OmTcData** structure. If the function returns an error, nothing will be written using the pointer.

Return value

The **OmTcData** structure is defined in omtcdata.h. The structure includes methods for extracting the timecode into various formats. The **OmTcData** structure returned by this function contains the time value but does not contain any user data. The time value is 00:00:00:00 for a clip that has not yet been recorded; the time value is available to this function within a few frames after recording is started. The value is 00:00:00:00 if the timecode source selected during the recording of the clip did not have valid timecode.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrClipSetStartTimecode, *OmPlrOpen*, *OmPlrClipGetInfo*

OmPlrClipSetStartTimecode

Description

Sets the timecode of the first frame in a clip.

```
OmPlrError OmPlrClipsetStartTimecode(
OmPlrHandle plrHandle,
const char *clipName,
OmTcData startTc;
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
clipName	Identifies the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see the *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
startTC	Identifies the first timecode for the given clip.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

See also

OmPlrClipGetStartTimecode, *OmPlrOpen*, *OmPlrClipGetInfo*

OmPlrClipGetTrackUserData

Description

Gets track clip user data.

```
OmPlrError OmPlrClipGetTrackUserData(
OmPlrHandle plrHandle,
const TCHAR *pClipName,
int trackNum,
const char *pKey,
unsigned char *pData,
uint dataSize,
uint *pRetDataSize);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipName	A pointer to a null-terminated string that gives the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see the *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
trackNum	Identifies the first timecode for the given clip.
pKey	A pointer to the name of the user data key.
pData	A pointer to the location of the returned user data.
dataSize	A numeric value that shows the size of the **pData** buffer (in bytes).
pRetDataSize	A numeric value that shows the size (in bytes) of the returned user data in the **pData** buffer.

Return value

Returns the user data stored under **pKey**. Tracks 1 to N are for the N video tracks. Tracks N+1 to N+M are for the M audio tracks. Use *OmPlrClipGetInfo* to find the number of video and audio tracks.

Remarks

Similar to *OmPlrClipGetUserData* except that it accesses track level user data as opposed to clip level user data.

See also

OmPlrOpen, *OmPlrClipGetInfo*, *OmPlrClipGetUserData*

OmPlrClipGetTrackUserDataAndKey

Description

Gets track clip user data by index.

```
OmPlrError OmPlrClipGetTrackUserDataAndKey(
OmPlrHandle plrHandle,
const TCHAR *pClipName,
int trackNum,
uint dataIndex,
char *pKey,
uint keySize,
unsigned char *data,
uint dataSize,
uint *retDataSize);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipName	A pointer to a null-terminated string that gives the name of the source clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see the *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
trackNum	Identifies the track in the clip. Tracks are consecutively numbered starting from 0. Track 0 is for the movie file. Tracks 1 to N are for the N video tracks. Tracks N+1 to N+M are for the M audio tracks.
dataIndex	Specifies which user data to obtain from a clip. Range is 0 to n -1.
pKey	A pointer to a buffer. Will be used for returning the user key bytes if the clip has a "n'th" set of user data. The returned bytes, if any, will be a proper C string with a terminating byte of 0.
keySize	A numeric value that shows the size of the **pData** buffer (in bytes).
data	A numeric value that shows the size (in bytes) of the returned user data in the **pData** buffer.
dataSize	A numeric value that shows the size (in bytes) of the above **data** buffer.
***retDataSize**	A numeric value that show the size (in bytes) of the user data returned.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Returns the n'th user data/key pair. Tracks 1 to N are for the N video tracks. Tracks N+1 to N+M are for the M audio tracks. Use *OmPlrClipGetInfo* to find the number of video and audio tracks.

Remarks

This is similar to *OmPlrClipGetUserDataAndKey* except that it accesses track level user data instead of accessing clip level user data.

See also

OmPlrOpen, *OmPlrClipGetUserDataAndKey*, *OmPlrClipGetInfo*

OmPlrClipGetUserData

Description

Gets user data from a clip that has been stored using a key.

```
OmPlrError OmPlrClipGetUserData(
```

```
                OmPlrHandle playerHandle,
                const TCHAR *pClipName,
                const char*pKey,
                unsigned char*pData,
                uint dataSize,
                uint *pRetDataSize);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipName	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
pKey	A pointer to a string that is the key for the desired data. This string will be matched to the key that was used for storing the user data; the match is case sensitive. Needs to be a proper "C" string with a byte of 00 as a terminator. Maximum length is **omPlrMaxKeyLen(64)**.
pData	A pointer to a buffer. Will be used for returning the user data bytes if a match is found to the key. The bytes can have any value, including 00. The data bytes will be returned in the same order as they were stored.
dataSize	Size of the buffer for holding the user data. Sets the upper limit on how much user data will fit into the buffer.
pRetDataSize	A required pointer used to return the actual number of user data bytes that were retrieved.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file playerdefs.h. The error codes are assigned in ranges. The range for player errors starts at PLAYER_ERROR_BASE (0x00009000).

A value of **omPlrInvalidClip** is returned if the clip could not be found in the current directory.

A value of **omPlrUserDataNotExist** is returned if no match was found for the Key in the indicated clip.

A value of **omPlrBufferTooSmall** is returned if the length of the user data exceeds the dataSize value.

Other error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrOpen, *OmPlrClipSetUserData*, *OmPlrClipGetUserDataAndKey*, *OmPlrClipSetDirectory*

OmPlrClipGetUserDataAndKey

Description

Gets user key and data from a clip using a key index.

```
OmPlrError OmPlrClipGetUserDataAndKey(
OmPlrHandle playerHandle,
const TCHAR *pClipName,
uint index,
char *pKey,
uint keySize,
unsigned char *pData,
uint dataSize,
uint *pRetDataSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipName	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see the *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
index	A number indicating which set of user key/data should be retrieved. The first set has an index of 0.
pKey	A pointer to a buffer. Will be used for returning the user key bytes if the clip has a "n'th" set of user data. The returned bytes, if any, will be a proper C string with a terminating byte of 0.
keySize	Size of the buffer for holding the user key. Sets the upper limit on the longest user key that will fit into the buffer.
pData	A pointer to a buffer. Will be used for returning the user data bytes if a match is found to the key. The bytes can have any value, including 00. The data bytes will be returned in the same order as they were stored.
dataSize	Size of the buffer for holding the user data. Sets the upper limit on how much user data will fit into the buffer.
pRetDataSize	A required pointer used to return the actual number of user data bytes that were retrieved.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file playerdefs.h. The error codes are assigned in ranges. The range for player errors starts at PLAYER_ERROR_BASE (0x00009000).

A value of **omPlrInvalidClip** is returned if the clip could not be found in the current directory.

A value of **omPlrUserDataNotExist** is returned if the index equals or exceeds the number of user key + data sets attached to the indicated clip.

A value of **omPlrBufferTooSmall** is returned if the length of the key data exceeds the **keySize** value.

A value of **omPlrBufferTooSmall** is returned if the length of the user data exceeds the dataSize value.

Other error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Typically the "0th" key is ovn_creation and was created by Spectrum firmware.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrOpen, *OmPlrClipSetUserData*, *OmPlrClipSetDirectory*, *OmPlrClipGetUserData*

A value of **omPlrInvalidClip** is returned if the clip could not be found in the current directory.

A value of **omPlrUserDataNotExist** is returned if the index equals or exceeds the number of user key + data sets attached to the indicated clip.

A value of **omPlrBufferTooSmall** is returned if the length of the key data exceeds the **keySize** value.

A value of **omPlrBufferTooSmall** is returned if the length of the user data exceeds the dataSize value.

Other error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Typically the "0th" key is ovn_creation and was created by Spectrum firmware.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrOpen, *OmPlrClipSetUserData*, *OmPlrClipSetDirectory*, *OmPlrClipGetUserData*

OmPlrClipInsertData

Description

Inserts data of type **dataType** into clip **pClipName**.

```
OmPlrError OmPlrClipInsertData(
OmPlrHandle plrHandle,
const TCHAR *pClipName,
uint startFrame,
uint numFrames,
OmPlrClipDataType dataType,
unsigned char *pData,
uint *pDataSize);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipName	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see the *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
startFrame	A numeric value that provides the frame number to start data insertion from the clip.
numFrames	A numeric value that determines the number of frames to insert from a clip. This value can range from 1 to 512.
dataType	Determines the type of data to insert as specified by enum **OmPlrClipDataType** in the **omplrdefs.h** file.
pData	A pointer to the buffer of data to insert. The amount of inserted data is the number of frames (**numFrames**) times the number of bytes per frame. For example, inserting 100 frames of CC data (**omPlrClipDataCcFrame**) is 100 x 4 = 400 bytes.
pDataSize	A pointer to the amount of data in the **pData** buffer. It must be set by the caller to point to the size (in bytes) of the **pData** buffer data. Upon return, the value will be updated to the amount of data inserted.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file playerdefs.h. The error codes are assigned in ranges. The range for player errors starts at PLAYER_ERROR_BASE (0x00009000).

Upon successful return, **pDataSize** returns the amount of data inserted (in bytes).

Remarks

This function can insert a number of frames worth of Close Caption (CC) data starting at a specified frame into a specified clip. Inserting CC data into an entire clip for any clip more than a few seconds long requires multiple calls to **OmPlrClipInsertData**.

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 CC data into a 10,000 frame long clip, make 50 calls to **OmPlrClipInsertData** each inserting 200 frames worth of CC starting at frame locations 0, 200, 400, 600... 9800.

OmPlrClipInsertData is declared in the **omplrclnt.h** file.

See also

OmPlrOpen, *OmPlrClipExtractData*

OmPlrClipRegisterCallbacks

Description

Registers a set of functions that should be called whenever a clip is added or deleted from the storage medium.

```
OmPlrError OmPlrClipRegisterCallbacks(
OmPlrHandle playerHandle,
Void (* pAddedCallback)(const TCHAR *Name, int param),
intaddedParam,
void (* pDeletedCallback)(const TCHAR *Name, int param),
intdeletedParam);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pAddedCallback	A pointer to a function that will be called whenever a clip is added to the storage medium. The arguments passed to the function will be a pointer to the clip name and a value of **addedParam**. Use a value of 00 for **AddedCallback** if you do not want to have callbacks for added clips.
addedParam	A numeric value that will be supplied to the **AddedCallback** function whenever it is called.
pDeletedCallback	A pointer to a function that will be called whenever a clip is deleted from the storage medium. The arguments passed to the function will be a pointer to the clip name and a value of **deletedParam**. Use a value of 00 for **DeletedCallback** if you do not want to have callbacks for deleted clips.
deletedParam	A numeric value that will be supplied to the **DeletedCallback** function whenever it is called.

Return value

A numeric value that defines success or error. A value of 0 indicates “all ok.” Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Establishes a thread in your platform that sleeps waiting for the MediaDirector to send notification of clip changes. Call this function with NULL for both function pointers in order to shut down the thread. Each **playerHandle** has its own set of callback functions; each playerHandle must set up its callbacks if it wants notification. Callback functions are shut down when **OmPlrClose** is called.

The callbacks are made for changes in the clip directory that was the “current clip directory” for **playerHandle** at the time that this function was called. If you call **OmPlrClipSetDirectory** after this, you may want to recall this function to change the callback watch to the new clip directory.

Callbacks will be effected by the current setting of the extension list. The current setting can be determined by the function **OmPlrClipGetExtList**. For instance, if the settings at the time of the call to the **OmPlrClipRegisterCallbacks** function were an extension list of simply ".mov" and a clip directory of "/FS1/myClips.dir", then callbacks will only happen for changes to .mov clips in the myClips directory. If you are interested in callbacks for .mxf clips, you need to make sure the extension list is set to ".mxf' before calling **OmPlrClipRegisterCallback.s** Note that the actual callback function only specifies the clip name; the directory and the extension are not given. Changes to the current extension list made after **OmPlrClipRegisterCallbacks** is used to establish the callbacks *do not* effect the clip types that is being checked by the callback mechanism.

Callbacks will occur for changes to a clip that match any of the extensions that were in the extension list at the time the function was called. For example, if the extension list at the time of calling this function was ".mxf.mov", then callbacks would occur for either type of clip. Note that the callback does not return the extension nor does it return the clip directory. The callback does return a value of your choice that you set during the call of the function **OmPlrClipRegisterCallbacks**.

This function can be used even if a NULL Player name was used to open the connection.

OmPlrClipRegisterWriteCallbacks

Description

Registers a set of functions that will be called whenever the movie file of a clip is opened with write permission, or is closed after having been opened with write permission.

```
OmPlrError OmPlrClipRegisterWriteCallbacks(
OmPlrHandle plrHandle,
void (*openForWriteCallback)(const TCHAR *name, int
param),
int openParam,
void (*closeForWriteCallback)(const TCHAR *name,int
param),
int closeParam);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
OpenForWriteCallback	A pointer to a function that will be called whenever a clip file is opened with write permissions. The arguments passed to the function will be a pointer to the clip name and a value of **openParam**. Use a value of 00 for **OpenForWriteCallback** if you do not want to have callbacks for clips opened for write.
openParam	A numeric value that will be supplied to the **OpenForWriteCallback** function whenever it is called.

Parameter	Description
CloseForWriteCallback	A pointer to a function that will be called whenever a clip file is closed after having been opened with write permissions. The arguments passed to the function will be a pointer to the clip name and a value of **closeParam**. Use a value of 00 for **CloseForWriteCallback** if you do not want to have callbacks for clips closed for write.
closeParam	A numeric value that will be supplied to the **CloseForWriteCallback** function whenever it is called.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Establishes a thread in your platform that sleeps waiting for the MediaDirector to send notification of file operations that open or close a file that has an extension of .mov where the file open operation was with write permission.

For instance, a **CloseForWrite** callback will occur when recording of a clip is finished.Call this function with NULL for both function pointers in order to shut down the thread. Each **playerHandle** has its own set of callback functions; each **playerHandle** must set up its callbacks if it wants notification. Callback functions are shut down when **OmPlrClose** is called.

The callbacks are made for file operations in the clip directory that was the "current clip directory" for **playerHandle** at the time that this function was called. If you call **OmPlrClipSetDirectory** after this, you may want to recall this function to change the callback watch to the new clip directory.

Callbacks will be effected by the current setting of the extension list. The current setting can be determined by the function **OmPlrClipGetExtList**. For instance, if the settings at the time of the call to the **OmPlrClipRegisterCallbacks** function were an extension list of simply ".mov" and a clip directory of "/FS1/myClips.dir", then callbacks will only happen for changes to .mov clips in the myClips directory. If you are interested in callbacks for .mxf clips, you need to make sure the extension list is set to ".mxf' before calling **OmPlrClipRegisterCallback.s** Note that the actual callback function only specifies the clip name; the directory and the extension are not given. Changes to the current extension list made after **OmPlrClipRegisterCallbacks** is used to establish the callbacks *do not* effect the clip types that is being checked by the callback mechanism. s

Clip files are files in the clip directory (set by **OmPlrClipSetDirector**) that have a clip extension (set by **OmPlrClipSetExtList**). Before changing the clip directory or clip extension list, callbacks should be disabled by specifying NULL callback function names. Re-enable after setting the clip directory and/or clip extension list

This function can be used even if a NULL Player name was used to open the connection.

Similar to *OmPlrClipRegisterCallbacks* except the callback conditions are different.

Notes about callbacks while a clip is being recorded

- An **OpenForWrite** callback will occur when a new clip is attached to the timeline with the **OmPlrAttach** command.

- A **CloseForWrite** callback will occur for a new clip when the recording of that clip is completed by use of **OmPlrStop**. If you are recording a set of new clips, there will be a **CloseForWrite** callback for a particular clip after the timeline position moves some minimal amount beyond the particular clip.
- If you add or change user data on a clip while the clip is being recorded (using **OmPlrClipSetUserData**), there will not be any additional **OpenForWrite** or **CloseForWrite** callbacks.
- If you alter the value of the clip **DefaultIn** while the clip is being recorded, that also will not trigger any additional **OpenForWrite** or **CloseForWrite** callbacks.

Notes about callbacks at other times

There are several situations that can trigger **OpenForWrite** and **CloseForWrite** callbacks after a clip has been recorded. The following examples of when callbacks are currently triggered are not exhaustive and are subject to change in the future:

- A set of **OpenForWrite** and **CloseForWrite** callbacks occurs if a clip file is created using the **OmPlrClipCopy** function. The command, **OmPlrClipRename**, may generate the **openForWrite** and **closeForWrite** callbacks. Using **OmPlrClipRename** to rename an existing clip in the directory will generate deleted and added callbacks.
- A set of callbacks occurs when user data is added or changed for a clip using the **OmPlrClipSetUserData** function.
- A set of **OpenForWrite** and **CloseForWrite** callbacks occur when the **OmPlrClipSetDefaultInOut** function is used for a clip.
- A set of **OpenForWrite** and **CloseForWrite** callbacks occur when a clip file is copied into a directory via Samba, FTP, or AFP mechanisms.
- Typically, a multiple sets of **OpenForWrite** and **CloseForWrite** callbacks occurs when Microsoft Windows Explorer is used to examine properties of a .mov file.

See also

OmPlrOpen, *OmPlrClipRegisterCallbacks*

OmPlrClipRename

Description

Used to change the name of an existing clip on the storage medium. Can also be used to move an existing clip and its media from one directory to another directory inside the same Spectrum MediaDirector.

```
OmPlrError OmPlrClipRename(
OmPlrHandle playerHandle,
const TCHAR *pOldName,
const TCHAR *pNewName);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function OmPlrOpen.
pOldName	A pointer to a null-terminated string that gives the current name of the clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. Maximum length of the string (including the terminator byte) is **omPlrMaxClipDirLen** (512). The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can also include a full path name; see *Clip Name as Function Argument*.
pNewName	A pointer to a null-terminated string that gives the new name for the clip. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. Maximum length of the string (including the terminator byte) is omPlrMaxClipDirLen (512). The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can also include a full path name; see *Clip Name as Function Argument*.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000). An error of **omPlrFailure** will be returned if an attempt is made to rename to a clip name that already exists.

Remarks

For both RENAME and MOVE operation

- The source directory can be supplied with the OldName argument and otherwise will be the "current clip directory". Similarly, the destination directory can be supplied with the NewName argument and otherwise will be the "current clip directory".
- A clip consists of a "wrapper file" and several media files that are kept in the media.dir subdirectory (see *Clip Function Discussion* for details). The .mov file contains the names of the associated media files; one set of media references in the .mov file uses a relative path.
- If the clip does not exist in the source directory, the operation will fail.
- If any Player has registered for clip deleted callbacks in the source clip directory, a clip rename or move operation will generate a "deleted clip" callback for the old name for that player.
- If any Player has registered for clip added callbacks in the destination clip directory, then a clip rename or move operation will generate a "added clip" callback for the new name for that player.
- This function can be used even if a NULL Player name was used to open the connection.

For RENAME operation

- A simple rename occurs if the source directory is the same as the destination directory. This rename function only changes the name of the wrapper file; the file names of the media files are not changed.
- If the clip is the current clip for some timeline, the timeline status will change to reflect the name change.
- The rename operation is allowed for a clip that is currently loaded into a Player. The rename operation is allowed even while a Player is playing or recording.
- A rename that has the same new name as the old name returns an error but is otherwise harmless.

For MOVE operation

- A move operation occurs if source directory is not the same as the destination directory.
- The move operation will place the wrapper file in the destination directory; the media files will be placed into the media.dir subdirectory of the destination directory. The media files being moved will be renamed using the new clip name and if necessary, a random suffix will be appended to make the name different from existing media files. The contents of the wrapper file will be rewritten so that it refers to the renamed media files. After a successful move operation, the wrapper file and the associated media files will no longer exist in the source clip directory.
- If the destination directory or the media.dir subdirectory do not exist, they will be created.
- If the **newName** duplicates the name of a .mov file that exists in the destination directory, then the operation will fail.
- A move operation can use the original clip name when moving a clip from source to destination directories.
- If the operation fails, the .mov file and the associated media file will remain in the source directory with the original name. But if the operation fails because some of the source media files are missing, the .mov file will already have been moved to the destination directory.
- The move operation does not create new media files; this has two implications. The first is that the operation is quite fast. Secondly, the operation will succeed even if the media store does not have enough free space to temporarily hold two copies of the clip.
- This API call is useful in restoring a clip from archive. Using CIFS/SMB or AFP, place the clip and media files into a directory on the MediaDirector's file system that is not currently used as a clip directory. This includes placing the media files in the correct relative location; normally this is in a directory named “media.dir” one level below the .mov file. Then use this API call to move the clip including media files into the desired working directory. The API will handle any media file name conflicts as described above.
- The move operation is allowed for a clip that is currently loaded into a Player. The timeline status will change to reflect the new name and location; see *OmPlrGetPlayerStatus*, *OmPlrGetClipName*, *OmPlrGetClipPath*. The move operation is allowed even while a Player is playing or recording the clip that is being moved.

NOTE: OmPlrClipRename preserves the extension of the source clip regardless of the extension list setting. For example, if the source clip argument of "fred" matches to a clip "fred.mov", and the destination clip is "sally", then the **OmPlrClipRename** operation will create a clip named "sally.mov". The extension of the source clip is also preserved when the source argument is a fully qualified clip such as "/FS1/clip.dir/fred.mov". Note that specifying a mismatched set of extensions such as a source of "FS1/clip.dir/fred.mov" and a destination of "/FS1/clip.dir/sally.mxf" is not supported.

See also

OmPlrClipSetDirectory

OmPlrClipSetDefaultInOut

Description

Sets the default in/out points for a clip as stored on the storage medium.

```
OmPlrError OmPlrClipSetDefaultInOut(
OmPlrHandle playerHandle,
const TCHAR *pClipName,
uint defaultIn,
uint defaultOut);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function OmPlrOpen.
pClipname	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy." The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
defaultIn	A numeric value that will be stored with the clip on the storage medium. The value identifies the first frame of the clip that will be seen whenever the clip is attached to the timeline using the argument of **omPlrClipDefaultIn**.
defaultOut	A numeric value that will be stored with the clip on the storage medium. The value identifies the frame after the last frame of the clip that will be seen whenever the clip is attached to the timeline using the argument of **omPlrClipDefaultOut**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Returns an error of **omPlrFailure** if the **defaultOut** value is less than or the same as the **defaultIn** value.

Returns an error of **omPlrFailure** if the clip does not exist.

Remarks

Typically, the **DefaultIn** and **DefaultOut** values are used to mark and remember the desirable portion of a clip. Special argument values can be used with the **OmPlrAttach** function that attaches a clip to the timeline; these special argument values will cause the Attach function to retrieve the **DefaultIn** and **DefaultOut** values from the clip information and to use these values when attaching the clip to the timeline.

A newly recorded clip will have **defaultIn** and **defaultOut** values that are the same as the start and end of the recorded material.

An error will be returned if the **defaultOut** value is less than the **defaultIn** value. The **defaultIn** and **defaultOut** values cannot be set outside the limits of the recorded material.

Changing the value of **defaultIn** or **defaultOut** does not affect the properties of any timeline to which the clip is already attached.

This function can be used even if a NULL Player name was used to open the connection.

CAUTION: The current behavior of this function is that **defaultOut** is set to the clip's **lastFrame** value if the **defaultOut** argument is larger than the lastFrame value. The behavior may change in future versions of the MediaDirector firmware. You are advised to use the **OmPlrClipGetInfo** function to determine the actual **firstFrame** and **lastFrame** numbers before calling this function.

See also

OmPlrOpen, *Clip Function Discussion*, *Timeline Function Discussion*, *OmPlrAttach*

OmPlrClipSetDirectory

Description

Sets the current clip directory name.

```
OmPlrError OmPlrClipSetDirectory(
OmPlrHandle playerHandle,
Const TCHAR *pClipDir);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function OmPlrOpen.
pClipDir	A pointer to a buffer that holds the string that is the new clip directory. name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The maximum string size is **omPlrMaxClipDirLen**, inclusive of the terminating '\0.'

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The path name should be an ABSOLUTE path name. Each instance of playerHandle has its own current clip directory. A default value is established on connection to the Player that is "/FILESYS0/clip.dir" if the primary file system on the MediaDirector is called "FILESYS0." The directory is used for any file system related functions, such as **OmPlrAttach**, if the clip name is a simple name like "Fred" rather than a fully qualified name such as "/FS1/clip.dir/Fred.mov".

Directory name is case sensitive. The separating slashes must be a forward slash (leaning to the right). A trailing slash is optional (won't hurt, won't help). Wildcard characters are not supported. The name can contain spaces. It cannot contain any of the following characters / \ : * ? < > |. The double quote character (") is also not allowed. The directory name also can not contain instances of "." (period) to indicate current directory or ".." to indicate one Up directory.

This directory will be the "base directory" for the Spectrum media for this Player. The directory must already exist. This function does not create directories. The Spectrum file system keeps the actual media files in a subdirectory of the base directory that is called media.dir. This function will create the media.dir subdirectory if that subdirectory did not already exist.

The constant **omPlrMaxClipDirLen** is defined in the file omplrdefs.h; its current value is 512, which includes the terminating '\0' character.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrOpen, *Clip Function Discussion*, *OmPlrClipGetDirectory*

OmPlrClipSetExtList

Description

Sets clip extension list for the MediaDirector.

```
OmPlrError OmPlrClipSetExtList(
OmPlrHandle plrHandle,
const TCHAR *pClipExtList);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function OmPlrOpen.
pClipExtList	A pointer to a buffer that holds the clip extension list.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The extension list is a concatenated list of filename extensions that are used as clip identifiers. For example: for QuickTime files and .dv files the list would look like ".mov.dv."

This clip extension list is an attribute of the network connection to the MediaDirector, not an attribute of the player. Clip names specified in other functions use this list of extensions.

See also

OmPlrOpen

OmPlrClipSetProtection

Description

Used to set or clear "protection" for a clip on the storage medium.

```
OmPlrError OmPlrClipSetProtection(
OmPlrHandle playerHandle,
const TCHAR *pClipName,
bool protection);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function OmPlrOpen.
pClipName	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy."as "haRRy."The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
protection	A true or false value. A clip becomes protected if this parameter is true. The clip becomes unprotected if this parameter is false. A false value is a numeric 0; any non-zero value is considered to be a true value.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

While protection is ON, the functions **OmPlrClipDelete** and **OmPlrClipSetDefaultInOut** will not work for the clip. The function **OmPlrClipRename** is still allowed to change the name of the clip or move the clip; the renamed clip will continue to be protected. And the function **OmPlrRecord** is still allowed to start a recording process for a new clip even if protection has been set for the clip. The default in and out points will be updated as the recording progresses.

A newly created clip has its protection set to OFF. The protection property applies to the clip as it is stored in the Spectrum file system. The protection, once set, will apply to all players; the protection will persist after this Player connection has been shut down. The Spectrum clip file system can be seen using Windows Explorer. The protection property is not visible from Windows Explorer but the protection property does prevent Windows Explorer from deleting the clip.

To inquire about whether a clip is protected, use the **OmPlrClipGetInfo** function and then examine the protection field in the returned **OmPlrClipInfo** structure.

The Spectrum clip file system can be seen using Windows Explorer. The protection property is not visible from Windows Explorer but the protection property does prevent Windows Explorer from deleting the .mov file that holds the pointers and properties for the clip. The media data is kept in separate files in a subdirectory of the directory that holds the .mov file; these media files are not protected from being deleted using Windows Explorer.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrOpen, *OmPlrClipSetDirectory*, *OmPlrClipGetInfo*

OmPlrClipSetTrackUserData

Description

Sets track clip user data.

```
OmPlrError OmPlrClipSetTrackUserData(
OmPlrHandle plrHandle,
    const TCHAR *pClipName,
    int trackNum,
    const char *pKey,
    unsigned char *pData,
    uint dataSize);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this Player. It was obtained using the function OmPlrOpen.
pClipName	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that “Harry” is not the same clip as “haRRy.”as “haRRy.”The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
trackNum	Identifies the track in the clip. Tracks are consecutively numbered starting from 0. Track 0 is for the movie file. Tracks 1 to N are for the N video tracks. Tracks N+1 to N+M are for the M audio tracks.
pKey	A pointer to a string that becomes the key for later retrieving the user data. Needs to be a proper “C” string with a byte of 00 as a terminator. Maximum length is **omPlrMaxKeyLen(64)**.

Parameter	Description
pData	A pointer to a string of arbitrary data bytes that will be stored with the clip. The bytes can have any value, including 00. They can later be retrieved and will be returned in the same order as stored.
dataSize	Size of the data string, in units of bytes. Maximum size is **omPlrMaxUserDataDataLen**, which is a very large number that is about 16,000.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This is similar to *OmPlrClipSetUserData* except that it accesses track level userdata as opposed to clip level user data.

The **pData** is stored under the pKey. Spectrum recommends using key strings that start with something unique to your company to avoid key collision. Specify zero length data to delete a user data key. Note that **pData** must not be NULL even if dataSize is 0, or the call will fail. Tracks 1 to N are for the N video tracks. Tracks N+1 to N+M are for the M audio tracks. Use *OmPlrClipGetInfo* to find the number of video and audio tracks.

See also

OmPlrOpen, *Clip Name as Function Argument*, *OmPlrClipGetInfo*, *OmPlrClipSetUserData*

OmPlrClipSetUserData

Description

Stores an arbitrary byte string under a key with a clip.

```
OmPlrError OmPlrClipSetUserData(
OmPlrHandle playerHandle,
const TCHAR *pClipName,
const char *pKey,
unsigned char *pData,
uint dataSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function OmPlrOpen.
pClipName	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy."as "haRRy."The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.

Parameter	Description
pKey	A pointer to a string that becomes the key for later retrieving the user data. Needs to be a proper "C" string with a byte of 00 as a terminator. Maximum length is **omPlrMaxKeyLen(64)**.
pData	A pointer to a string of arbitrary data bytes that will be stored with the clip. The bytes can have any value, including 00. They can later be retrieved and will be returned in the same order as stored.
dataSize	Size of the data string, in units of bytes. Maximum size is **omPlrMaxUserDataDataLen**, which is a very large number that is about 16,000.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The clip must already exist. Spectrum recommends using key strings that start with something unique to your company to avoid key collision. Specify zero length data to delete any existing user data that was previously set with this clip using this key.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrOpen, *OmPlrClipGetUserData*, *OmPlrClipSetDirectory*

OmPlrClipWhereRecording

Description

Returns the serial number of the server recoding the specified clip.

```
OmPlrError OmPlrClipWhereRecording(
OmPlrHandle plrHandle,
const char *pClipName,
char *pSerialNumber,
uint serialNumberSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
pClipName	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy."as "haRRy."The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.

Parameter	Description
pSerialNumber	A pointer to a buffer that stores returned serial numbers.
serialNumberSize	A numeric value which shows the size of the buffer that stores returned serial numbers.

Return value

A numeric value that defines success or error. A value of 0 indicates “all ok.” Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The clip may not yet be recording even when it is reported as recording; it may only be queued up for recording. In addition, the clip may never be recorded if the player is stopped before the record begins. Returns a zero length string if not recording.

See also

OmPlrOpen, *OmPlrClipGetUserData*, *OmPlrClipSetDirectory*

Chapter 2
Player Motion Functions

This chapter provides Spectrum API Functions related to the manipulation of Players. The following functions are included:

Function	Description
OmPlrCuePlay	Prepare Player to play out clip.
OmPlrCueRecord	Prepare Player to record clip.
OmPlrCueRecord1	Prepares a Player to record a clip and specifies the record options.
OmPlrGoToTimecode	Change position in the current clip so that the internal timecode reader matches the target time.
OmPlrPlay	Set the timeline in motion playing at the given rate.
OmPlrPlayAt	Set timeline in motion playing at the given rate – at some particular time in the future.
OmPlrPlayDelay	Set the timeline in motion playing at the given rate after a fixed delay from "now".
OmPlrRecord	Set the timeline in motion recording incoming material.
OmPlrRecordAt	Set timeline in motion recording – at some particular future time.
OmPlrRecordDelay	Set timeline in motion recording – after a fixed delay from "now."
OmPlrSetPos()	Set the current position on the timeline ("goto" timeline position).
OmPlrSetPosD	Set the current position on the timeline ("goto" timeline position).
OmPlrStep()	Step the timeline by a signed N frames.
OmPlrStepD	Step the timeline by a signed N frames.
OmPlrStop	Stop the timeline motion.

Player Function Discussion

The "Player" is a software object inside the Spectrum MediaDirector It is responsible for pushing the media out onto the Spectrum Spectrum System. A Player can be commanded to play and to stop; you can load clips and eject clips. A Player can also turn things around to record media and create new clips. The behavior of a Player "feels" like a VTR, but it is much more than a VTR. A Player is very good at playing back sets of clips in a back-to-back fashion. It can quickly load clips and can jump almost instantly from one position to another on a clip – or on a set of clips. It can also record a set of clips.

The Player is created and configured using the Spectrum SystemManager. During the configuration, you will decide:

- Types of media supported (e.g., DV or MPEG, + AES + VBI)
- Record or playback (or both)
- Frame rate, PAL or NTSC (Preference for drop frame timemode in NTSC)

- Name of the Player
- Media Wrapper Format - QuickTime (**.mov**) or Material eXchange Format (**.mxf**)

A MediaDirector can have many Players configured. A Player can be configured but not active. The limit on the number of active Players is a function of how much bandwidth is required by each active Player.

Each Player exists inside its own particular MediaDirector; the API function that opens the connection to a Player requires the MediaDirector as an argument. The Player can only be controlled by functions sent to the IP address of its particular MediaDirector; a Player can only work with media that is part of the file system seen by its particular MediaDirector. A particular MediaDirector can have Players defined for a mix of frame rates; for instance, it can have some Players defined for the PAL frame rate and other Players defined for the NTSC frame rate. A particular MediaDirector can have Players defined for a mix of media types; for instance, some Players could be configured to handle DV media while others are defined to handle MPEG media.

A Player can only play the type of media for which it has been configured. On the other hand, you can load any clip onto a Player. [There is an exception — the Player will refuse to load a clip that was made at a different frame rate from the Player's configuration.] Thus, it is OK to load a clip that has both video and audio onto a Player that has been configured to only handle only video; but in this example, when the clip is played, the audio information will be ignored. Similarly, you are allowed to load a DV-format video clip onto a Player that has been configured to handle SDI format (but not DV format); but when you play the clip, the output will be black since there is no SDI media on the clip.

The Player provides the means to create new clips. Clips are created by recording them. This is a real time 1X process that captures the incoming media into a file. A Player is not an editing device. When it records a new clip, it records on all the tracks for which it has been configured. And once a clip has been recorded, additional material can not be added to the clip (neither appended nor inserted). And material can't be deleted from part of a clip once it has been recorded; the only file operations allowed are creating a clip, renaming a clip and deleting of a clip.

Much of the Player's function set deals with the timeline. Each Player has its own timeline. A clip is played by attaching it to the timeline and then playing the timeline.

A clip is created by the act of attaching a non-existent clip to the timeline. See *Timeline Function Discussion* for a detailed discussion of the timeline concept. The first part of the Timeline Function Discussion covers attaching clips to the timeline and other related clip manipulations as objects that are attached to the timeline. The motion controlling functions have been listed under this "Player Functions" category. These motion controlling functions control the motion of the timeline. Normally you will have clips attached to the timeline before putting the timeline into motion but that is not required.

Your software calls API functions when it wants to control the Spectrum Player. The first step is to call the **OmPlrOpen** function that "opens" the Player. This "open" function establishes a structure inside the computer, makes a connection across the Ethernet to the requested MediaDirector, and returns a handle that you will use in future function calls that use this connection. Each connection will only control a single Player at a time; you will open additional connections to control additional Players. You can have more than one connection to a particular Player in a MediaDirector; for instance, one connection might be used for sending functions while another connection is used to gather status. And other applications in other computers can also be controlling the Player. If you have multiple threads in your application, you can carefully share a connection between threads but you may find it easier to just have each thread open its own connection; a connection can only be shared between threads if you implement some scheme such as a mutex to insure that two threads do not make function calls at the same time for the same connection.

When you open a connection, you will specify a Player as well as the MediaDirector. For instance, you would specify the Player if your intent is to attach clips to the timeline and play them. But some of the API function calls deal with the file system or with general MediaDirector properties. For these functions you have a choice — you can call these functions with a connection that was made to a particular Player, or you can also use a connection that was opened with a NULL Player. The NULL Player connection is handy if you do not know the names of any Players; it is also handy if you only care about file data and do not want to be bothered with Player names. A NULL Player connection is created by passing a pointer value of 0 for the Player name argument when calling the **OmPlrOpen** function.

The Ethernet connection between your computer and the MediaDirector is created by code in the Spectrum supplied omplrlib.dll. The Ethernet connection typically uses TCP/IP; UDP/IP is an alternative. The connections are handled by an **Open Network Connection** (ONC) **Remote Procedure Call** (RPC) package that is bundled into the Spectrum supplied **omplrlib.dll**. The ONCRPC package is freeware based on code developed originally by Sun. Refer to *ONC RPC Information* for more details about this package and its license.

Since the connection uses RPC as the underlying control mechanism, the function calls from your software will block until a reply has been received from the MediaDirector (or until the timeout of 3 seconds has expired). For instance, a request for status will not complete until the status data has been returned from the MediaDirector. Most functions will return almost instantly. The **OmPlrCuePlay** and **OmPlrCueRecord** functions will block until the MediaDirector is ready to play or record; these functions can sometimes block for several seconds and have a timeout value of 15 seconds.

OmPlrCuePlay

Description

Prepares a Player to play out a clip.

```
OmPlrError OmPlrCuePlay(
OmPlrHandle playerHandle,
double rate);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
rate	A signed floating point number that gives the desired speed. Use the value of 1.0 for 1X forward play.

Return value

A numeric value that defines success or error. A value of 0 indicates “all ok.” Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

The function will fail if the Player is not configured for playback.

Remarks

Readies a Player to start playing the clips on its timeline. Puts the timeline into its omPlrCueplay state with no motion (although the Player status reports the desired rate). Does not alter the timeline position.

This function is intended to be used as the last step of getting the Player ready to play at the requested rate. The function blocks until the Player is ready to play at the requested rate (or has returned an error condition). Other sequences of functions may not play all frames correctly if not enough time is allowed before the play function.

The rate argument is 64 bits that are split into a 32 bit integral part and a 32 bit fractional part. The compiler can handle creating these numbers if given inputs such as "1.5" or "100.0." More details about rate are in the description of the **OmPlrPlay** function

If the Player is in a recording or Cue record state, a stop function will need to be issued before this **OmPlrCuePlay** function will work.

The Spectrum MediaDirector is broadcasting media data onto the Spectrum network while in the **omPlrCuePlay** state. It will be repeatedly sending out the media frame that is at the current position of the Player's timeline. In the typical case of using Spectrum MediaPorts, you will see a still picture of the first expected frame.

CAUTION: When you are working with a clip that is currently being recorded, you should insure that a minimum number of the frames needed for playback have already been recorded before issuing the OmPlrCuePlay command; otherwise some black frames will be seen in the playback. The minimum number is 10 seconds if the playback is on the same host as the recording; the minimum number is 40 seconds if the recording is being done on a different host from the playback. The bit notReadyForPlay in the information returned by OmPlrClipGetInfo can be examined to see if enough frames have been recorded for the case of playing the clip from the firstFrame; the bit will change from 1 to 0 when it is ok to issue the OmPlrCuePlay command.

See also

Timeline Function Discussion, *OmPlrOpen*, *OmPlrPlay*

OmPlrCueRecord

Description

Prepares a Player to record a clip.

```
OmPlrError OmPlrCueRecord(
OmPlrHandle playerHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. Error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000). The function fails if the Player is not configured to record.

Remarks

Readies a Player to start recording a clip (or a series of clips). Puts the timeline into its **omPlrCuerecord** state with no motion (although the Player status reports a rate of 1.0). Does not alter the timeline position. This function will only work if the previous state of the Player's timeline was **omPlrStop**. This function will fail if there are non-empty clips attached to the timeline. This function will fail if the Player is not configured to enable recording.

This function is intended to be used as the last step of getting the Player ready to record. This function does not declare itself finished until after the Player is ready to record (or has returned an error condition). The function includes a check for valid incoming data. If no data is seen before a timeout, the function aborts with an error condition. Timeout is currently defined as 2 seconds.

Only empty clips can be recorded; the empty clip(s) is created using **OmPlrAttach**. An empty clip will have a **LastFrame** that is the same as its **FirstFrame**. The expected record duration is established by the difference between the IN and OUT arguments that were used with the **OmPlrAttach** function.

If the Player is not in the correct state, a stop function will need to be issued before this **OmPlrCueRecord** function will work. If the timeline has clips attached to it that have already been recorded, then the clips will need to be detached before the **OmPlrCueRecord** function will work. Notice that the **OmPlrCueRecord** function can be used on an empty timeline.

Once the Player's timeline state has become omPlrCuerecord, most functions are locked out. The functions still accepted are **OmPlrStop**, **OmPlrCueRecord**, **OmPlrRecord**, **OmPlrRecordAt**, and **OmPlrRecordDelay**. Some timeline releated functions are still accepted; e.g., **OmPlrSetClipData** can be used to change IN and OUT points. And functions that set timeline min, max, and current position can be used. The **OmPlrDetach** and **OmPlrDetachAllClips** functions are also locked out – but only if the **pos** parameter of the timeline is the same as the **minPos** parameter of the timeline.

See also

Timeline Function Discussion, *Clip Function Discussion*, *OmPlrRecord*, *OmPlrAttach*, *OmPlrOpen*

OmPlrCueRecord1

Description

Prepares a Player to record a clip and specifies the record options.

```
OmPlrError OmPlrCueRecord1(
OmPlrHandle plrHandle),
uint recordOptions );
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
recordOptions	This value is defined in omplrdefs.h. The only defined option is **OmPlrRecordStopAtMaxPos**. When this option is set, a Player stops recording and enters a stopped state when the Player position reaches the maximum position as specified by **OmPlrSetMaxPos**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. Error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000). The function fails if the Player is not configured to record.

Remarks

Readies a Player to start recording a clip (or a series of clips). Puts the timeline into its **omPlrCuerecord** state with no motion (although the Player status reports a rate of 1.0). Does not alter the timeline position. This function will only work if the previous state of the Player's timeline was **omPlrStop**. This function will fail if there are non-empty clips attached to the timeline. This function will fail if the Player is not configured to enable recording.

This function is intended to be used as the last step of getting the Player ready to record. This function does not declare itself finished until after the Player is ready to record (or has returned an error condition). The function includes a check for valid incoming data. If no data is seen before a timeout, the function aborts with an error condition. Timeout is currently defined as 2 seconds.

Only empty clips can be recorded; the empty clip(s) is created using **OmPlrAttach**. An empty clip will have a **LastFrame** that is the same as its **FirstFrame**. The expected record duration is established by the difference between the IN and OUT arguments that were used with the **OmPlrAttach** function.

If the Player is not in the correct state, a stop function will need to be issued before this **OmPlrCueRecord1** function will work. If the timeline has clips attached to it that have already been recorded, then the clips will need to be detached before the **OmPlrCueRecord1** function will work. Notice that the **OmPlrCueRecord1** function can be used on an empty timeline.

Once the Player's timeline state has become omPlrCueRecord1, most functions are locked out. The functions still accepted are **OmPlrStop**, **OmPlrCueRecord**, **OmPlrRecord**, **OmPlrRecordAt**, and **OmPlrRecordDelay**. Some timeline related functions are still accepted; e.g., **OmPlrSetClipData** can be used to change IN and OUT points. And functions that set timeline min, max, and current position can be used. The **OmPlrDetach** and **OmPlrDetachAllClips** functions are also locked out – but only if the **pos** parameter of the timeline is the same as the **minPos** parameter of the timeline.

See also

Timeline Function Discussion, *Clip Function Discussion*, *OmPlrRecord*, *OmPlrAttach*, *OmPlrOpen*

OmPlrGoToTimecode

Description

Changes position in the current clip so that the internal timecode reader matches the target time.

```
OmPlrError OmPlrGoToTimecode(
OmPlrHandle playerHandle,
OmTcData targetTime,
uint timeoutMsecs);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
targetTime	A **OmTcData** structure that contains the target position. The user bits in the structure are ignored. The structure is defined in the file omTcData.h; see *OmPlrGetTime* for a discussion of the structure.
timeoutMsecs	This value sets a limit for how long the MediaDirector will try to find the target position. The value is in units of milli-seconds.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This function positions the clip to a particular timecode that is read from the clip. It does the equivalent of the **OmPlrGetTime** function to examine the current **videoTc** of the Player. This function only works if the Player state is **CuePlay**, or Play with a speed of 0; otherwise it returns an **omPlrWrongState** error. After a successful positioning, this function waits until the Player device has had a chance to cue before returning, so it is not necessary to send **cuePlay** after this completes.

This function works best if the initial position and the **targetTime** all lie within the same clip; in some cases, the function will also work if multiple clips are loaded on the timeline and the initial position and **targetTime** are in two different clips. The function will never go beyond the timeline settings for minimum and maximum position. The function has a +/- 12 hours test so that it can handle the situation of an initial position on one side of midnight and the targetTime on the other side.

This function takes the frame rate from the device configuration. For NTSC, the function uses the equivalent of **OmPlrGetTime** to examine the Player's current **videoTc** to determine if the clip time is in drop frame. The user bit information of the target time is ignored. In addition to the timeout limit, this function has a small fixed limit on how many iterations it makes while trying to find the correct position; this secondary limit blocks thrashing around if the clip's target time does not exist.

See also

OmPlrGetTime, *OmPlrGetTcgInsertion*, *OmPlrSetTcgMode*, *Timecode Discussion*

OmPlrPlay

Description

Sets the timeline in motion playing at the given rate.

```
OmPlrError OmPlrPlay(
OmPlrHandle playerHandle,
double rate);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
rate	A numeric value that sets the timeline motion rate. The rate is a signed floating point number that gives the desired speed; rate is passed in a 64bit value. The value of 1.0 indicates 1X forward play. Values must be within the inclusive bounds of +128.0 to −128.0.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

The function will return an error if the rate exceeds the limits.

Remarks

This **OmPlrPlay** function will return an error if the current Player motion state is not omPlrPlay or omPlrCuepay; if the Player is not in the correct state, then first use the **OmPlrCuePlay** function (and you may need to use **OmPlrStop** before that).

This function causes the Player state to become **omPlrPlay**. The Player's timeline rate will be set by the rate argument. The rate argument can be any valid value – typically 1X forward but can also be 0.0 for still, or some reverse speed or some very fast forward speed. The rate will retain its value even if later the timeline is stopped or unloaded. While in this state of omPlrPlay, the Player will broadcast media frames onto the Spectrum 1394 network; if the timeline position goes beyond the limits of attached clips, then the Player will be sending black frames. A frame of media is broadcast at a steady 1X frame rate; play rates of other than 1X speed are obtained by repeating frames for slower rates and skipping frames for higher rates.

The play motion will start from the current timeline position, which is the Player's **pos** parameter. The play motion will continue until some limit is reached. If looping is enabled, then the limits are set by the Player's **loopMin** and **loopMax** parameters. The limits for non-looping operation are set by the Player's **minPos** and **maxPos** parameters; the minPos parameter is an inclusive limit while the **maxPos** is an exclusive limit. When the limit is reached during non-looping operation, the last frame and the rate will be unchanged.

In many cases you will set up the timeline position, the minimum timeline position, and the maximum timeline position before issuing **OmPlrPlay**. Use of the **OmPlrCuePlay** function as the last setup function before the **OmPlrPlay** function call will guarantee that the MediaDirector logic was given enough time to prepare the media frames needed for the play operation; the same rate would be used with the **OmPlrCuePlay** function that will be used with the **OmPlrPlay** function. The present values of these Player parameters may be obtained using the **OmPlrGetPlayerStatus** function.

After the **OmPlrPlay** function has been sent, play motion will start as soon as possible. The first frame will show up at the output after some play latency frames of delay. Play latency is deterministic and repeatable for a given set of hardware and firmware (if **OmPlrCuePlay** is used or enough delay is allowed after the last setup function). In the typical case of using an Spectrum MediaPorts for the output device, the play latency is about 2 to 10 frames and depends on the type of MediaPorts. The **OmPlrPlayAt** and the **OmPlrPlayDelay** functions can be used if more control is needed over when the first frame will appear at the output.

This function will fail if the Player is not configured to enable playback.

Note that the Player is happy to play even if no clips are attached. It is also happy to play beyond the limits of any attached clips.

See also

OmPlrCuePlay, *OmPlrPlayAt*, *OmPlrPlayDelay*, *OmPlrGetPlayerStatus*, *OmPlrSetPos()*, *OmPlrSetMinPos*, *OmPlrSetMaxPos*, *OmPlrLoop*

OmPlrPlayAt

Description

Sets the timeline in motion playing at the given rate – at some particular time in the future.

```
OmPlrError OmPlrPlayAt(
OmPlrHandle playerHandle,
uint startAtSysFrame,
double rate);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
startAtSysFrame	A numeric value that sets when the Player should go into play. The value is referenced to the system frame counter, which is an arbitrary counter of frames located inside the MediaDirector.
rate	A numeric value that sets the timeline motion rate. The rate is a signed floating point number that gives the desired speed; rate is passed in a 64bit value. The value of 1.0 indicates 1X forward play. Values must be within the inclusive bounds of +128.0 to –128.0.

Return value

A numeric value that defines success or error. A value of 0 indicates “all ok.” Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The remarks that were given for the **OmPlrPlay** function also apply to this function. Those remarks covered such things as rate and limits for the timeline motion.

This function will fail if the Player’s timeline state was not **omPlrCuePlay** or **omPlrPlay**.

The **startAtSysFrame** value is referenced to the value of the system frame counter. The system frame counter is a counter of frames that is located inside the MediaDirector. The system frame counter will count at the frame rate of the Player and the value of the system frame counter will depend on the frame rate of the player. The counter is set to 00 when the MediaDirector is reset. It will take a little over two years before it starts to wrap. You can determine the current relationship between the system frame counter and the houseVITC timecode by using the **OmPlrGetTime** function with the connection for this player. The system frame counter will count at the rate of the MediaDirector analog reference, which may not be at the frame rate of the Player.

This function causes the start of the timeline play motion to occur so that the first frame appears at the output at the requested system frame count value. If the **startAtSysFrame** value is in the past, then the action will be the same as if the function had been **OmPlrPlay**. Typically the **OmPlrPlayAt** function will be used to put the timeline into 1X forward motion, but it can be used to establish any speed. In the general case, the **OmPlrPlayAt** function provides a means for providing a scheduled speed change.

This function will be rejected if the Player state is not already **omPlrPlay** or **omPlrCueplay**. The status values for play state and the rate will not change until the delay time has expired and the playing motion has started.

Only one "pending play" condition can exist for a particular Player at any one time. And there are numerous ways to cancel a "pending play" condition. The various conditions include:

- If a new **OmPlrPlayAt** function is used after an earlier function call, then the **startAtSysFrame** of the new function will replace the start value that had been sent with the first function call.
- A "pending play" condition will be updated by a subsequent **OmPlrPlayDelay** function call.
- A "pending play" condition will be cancelled by a **OmPlrStop**, **OmPlrCuePlay**, or **OmPlrStep** function call.
- A **OmPlrPlay** function call will cancel the "pending play" condition only if the Player state had been **omPlrCueplay**.
- **OmPlrDetach** and **OmPlrDetachAllClips** do not cancel a "pending play" condition.
- **OmPlrClose** doesn't abort a "pending play" condition on the Player.

CAUTION: This function establishes a "pending play" condition. It is advisable to use a **OmPlrStop** function as redundant insurance to remove any lingering "pending play" condition when starting a new sequence on the timeline. Note that if there are multiple connections to one Player, then any of the connections can send functions that will cause the pending OmPlrPlayAt condition to be changed.

See also

OmPlrCuePlay, *OmPlrPlay*, *OmPlrGetTime*

OmPlrPlayDelay

Description

Sets the timeline in motion playing at the given rate after a fixed delay from "now."

```
OmPlrError OmPlrPlayDelay(
OmPlrHandle playerHandle,
uint startDelay,
double rate);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
startDelay	A numeric value that sets how long before the Player should go into play (or have the speed changed if the Player is already in omPlrPlay state). The value is in units of frames that tick off at the frame rate used by the Player.
rate	A numeric value that sets the timeline motion rate. The rate is a signed floating point number that gives the desired speed; rate is passed in a 64bit value. The value of 1.0 indicates 1X forward play. Values must be within the inclusive bounds of +128.0 to −128.0.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The remarks that were given for the **OmPlrPlay** function also apply to this function. Those remarks covered such things as rate and limits for the timeline motion.

This function will fail if the Player's timeline state was not **omPlrCueplay** or **omPlrPlay**.

This function causes the timeline play motion to start so that the first frame appears at the output after the requested frame delay from now. If the **startDelay** number is less than the play latency, then the function will have the same effect as the **OmPlrPlay** function. Typically the **OmPlrPlayDelay** function will be used to put the timeline into 1X forward motion, but it can be used to establish any speed. In the general case, the **OmPlrPlayAt** function provides a means for providing a scheduled speed change.

This function will be rejected if the Player state is not already **omPlrCueplay** or **omPlrPlay**. The status values for play state and the rate will not change until the delay time has expired and the playing motion has started.

Only one "pending play" condition can exist for a particular Player at any one time. There are numerous ways to cancel a "pending play" condition. The various conditions include:

- A new **OmPlrPlayDelay** function will always update any "pending play" condition; the delay of the new function will be used to create a new start value that replaces the one associated with any earlier function.
- A "pending play" condition will be updated by a subsequent **OmPlrPlayAt** function call.
- A "pending play" condition will be cancelled by a **OmPlrStop**, **OmPlrCuePlay**, or **OmPlrStep** function call.
- A **OmPlrPlay** function call will cancel the "pending play" condition only if the Player state had been **omPlrCueplay**.
- **OmPlrDetach** doesn't cancel a "pending play" condition.
- **OmPlrClose** doesn't abort a "pending play" condition on the Player.

CAUTION: This function establishes a "pending play" condition. It is advisable to use a **OmPlrStop** function as redundant insurance to remove any lingering "pending play" condition when starting a new sequence on the timeline. Note that if there are multiple connections to one Player, then any of the connections can send functions that will cause the pending **OmPlrPlayAt** condition to be changed.

See also

OmPlrCuePlay, *OmPlrPlay*, *OmPlrPlayAt*

OmPlrRecord

Description

Sets the timeline in motion recording incoming material.

```
OmPlrError OmPlrRecord(
OmPlrHandle playerHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The **OmPlrRecord** function will be accepted only if the current Player motion state is omPlrCuerecord or omPlrRecord. Issuing the **OmPlrRecord** command while the Player is already recording has no effect. This function will fail if the Player is not configured to enable recording.

The recording process can only happen for "empty clips." An "empty clip" is created by using the **OmPlrAttach** function with a clip name that doesn't match any existing clip. It is not possible to over record onto existing clips, although you can delete an existing clip and then create a new empty clip of the same name. See *Clip Function Discussion* for more details on clip creation.

The In and Out points of the clip as attached to the Player's timeline determine the maximum amount of material that will be recorded into the clip. The In and Out points were set as the clip was attached; note that if you attach using default In and Out, the clip will be attached with an In and Out of 00. After the clip is attached, you can use the **OmPlrSetClipData** function to change the In and Out points. Typically the In point will be set to 0, but any positive number can be used. An error will occur if Out is smaller than In; the clip will stay empty if Out is the same as In. In order to be able to record into the clip, the timeline's **pos**, **minPos** and **maxPos** parameters also need to be set

The **OmPlrRecord** function causes the timeline's state to become **omPlrRecord**. The first frame captured will be the frame following the frame in which the function arrived. The recording process will happen on all tracks that have been configured for this Player. The recording process starts from the current position of the timeline, as given by its **pos** parameter. If that position doesn't match the attach point of some clip, then the **defaultIn** of the clip will be automatically adjusted to be clip frame number that corresponds to the starting timeline position. In this case, the effect is as if the clip was attached with a different In point that matched the starting point on the timeline.

The recording of new material will continue until the timeline has reached its **maxPos** value. When the timeline's **maxPos** limt has been reached, the timeline state remains **omPlrRecord** but the pos parameter of the timeline stops changing. The **OmPlrStop** function is used to exit the recording state. The **OmPlrStop** function can also be used before the **maxPos** limit is reached if you want to terminate recording earlier; in that case, the last frame recorded will be the frame during which the **OmPlrStop** function was called.

The clip information that is kept as part of the Spectrum file system is constantly being updated while clip recording is in process. The information can be obtained with the function **OmPlrGetInfo**. The **firstFrame** value and the **defaultIn** value will not be changing. The **defaultIn** value of the clipInfo will be the first media frame recorded. The **defaultOut** value and the lastFrame value will be increasing as the recording proceeds; they will be the same value and will indicate the last frame recorded. The duration of the recording will be defined by "–**lastFrame** - **firstFrame**". If you are using Spectrum MediaPorts, the MediaPorts will drive its output signal from the input signal while the Player state is *omPlrRecord*.

In the typical case, you will prepare for recording by issuing a **OmPlrDetachAllClips** function, a **OmPlrStop** function, and then using the **OmPlrAttach** function to attach a new clip to the timeline. Then use functions to set the timeline position, timeline **minPos**, and timeline **maxPos** parameters. Finally use the **OmPlrCueRecord** function. The Player will then be waiting for the **OmPlrRecord** function to start recording.

Typically one clip is recorded at a time. But you can also attach a set of new clips and then do a continuous record into the set of them. Or you could start recording one clip and then load a second new clip before the recording of the first was finished. In this case, the recording process would record into the full length of the first clip and then start putting the material into the second clip. The second clip needs to be loaded at least a few seconds before the recording of the first clip is completed. You can also adjust the out points of clips if the adjustment is done at least a few seconds before the recording process reaches the end of that particular clip. You can use the **OmPlrDetach** function to remove the clips that have already been recorded see the caution in the **OmPlrDetach** remrks about waiting a few seconds before detaching in this situation of recording a series of clips.

Note that the Player will record even if no clips are attached. It will also record beyond the end of any attached clips.

See also

OmPlrCueRecord, *OmPlrRecordAt*, *OmPlrRecordDelay*, *OmPlrSetPos()*, *OmPlrSetMaxPos*, *OmPlrSetMinPos*

OmPlrRecordAt

Description

Sets the timeline in motion recording – at some particular time in the future.

```
OmPlrError OmPlrRecordAt(
OmPlrHandle playerHandle,
uint startAtSysFrame);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
startAtSysFrame	This value sets when the Player should start recording. The value is referenced to the system frame counter, which is an arbitrary counter of frames located inside the MediaDirector.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This function is similar to the **OmPlrRecord** function except, that it doesn't start the recording process until some future time. Most of the remarks for the **OmPlrRecord** function also apply to this function. The function will only be accepted if the Player state had been **omPlrCuerecord** or **omPlrRecord**. A **OmPlrRecordAt** function that is executed while already recording is quietly ignored.

The **startAtSysFrame** value is referenced to the value of the system frame counter. The system frame counter is a counter of frames that is located inside the MediaDirector. The system frame counter will count at the frame rate of the Player and the value of the system frame counter will depend on the frame rate of the player. The counter is set to 00 when the MediaDirector is reset. It will take a little over two years before it starts to wrap. You can determine the current relationship between the system frame counter and the house VITC timecode by using the **OmPlrGetTime** function with the connection for this player. The system frame counter will count at the rate of the MediaDirector analog reference, which may not be at the frame rate of the Player.

This function creates a "pending record" condition. As soon as the function is accepted the timeline state will change to **omPlrRecord** at a rate of 1.00, even though recording hasn't actually started yet. The function causes the timeline play motion to start so that the first frame recorded into the clip appeared at the input at the requested system frame count value. If the **startAtSysFrame** value is in the past, then the action will be the same as if the function **OmPlrRecord** had been used; the function returns an "all ok" value if the requested start time is in the past.

Only one "pending record" condition can be pending for a particular Player at any one time. Immediately after a **OmPlrRecordAt** function call the Player state changes from **omPlrCuerecord** to **omPlrRecord** – so most other functions are locked out. A **OmPlrRecord** command sent during the "pending record" interval will be quietly ignored; it will not start the recording. Sending a **OmPlrRecordDelay** or a new **OmPlrRecordAt** will update the "pending record" condition with the new information.

NOTE: OmPlrClose does not cancel a "pending record" condition. If the OmPlrDetach or OmPlrDetachAllClips functions can be used, they also do not cancel a "pending record" condition.

CAUTION: This function establishes a "pending play" condition. It is advisable to use a **OmPlrStop** function as redundant insurance to remove any lingering "pending play" condition when starting a new sequence on the timeline. Note that if there are multiple connections to one Player, then any of the connections can send functions that will cause the pending **OmPlrPlayAt** condition to be changed.

See also

Clip Function Discussion OmPlrCueRecord, *OmPlrRecord*, *OmPlrRecordDelay*, *OmPlrGetTime*

OmPlrRecordDelay

Description

Sets the timeline in motion recording – after a fixed delay from "now."

```
OmPlrError OmPlrRecordDelay(
OmPlrHandle playerHandle,
uint startDelay);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
startDelay	A numeric value that sets how long before the Player should begin the recording process. The value is in units of frames that tick off at the frame rate used by the Player.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This function is similar to the **OmPlrRecord** function except that it doesn't start the recording process until some future time. Most of the remarks for the **OmPlrRecord** function also apply to this function. The function will only be accepted if the Player state had been **omPlrCuerecord** or **omPlrRecord**. An **OmPlrRecordDelay** function that is executed while already recording is quietly ignored.

This function creates a "pending record" condition. The function causes the timeline play motion to start so that the first frame recorded into the clip appears at the input after the requested delay. As soon as the function is accepted the timeline state will change to **omPlrRecord** at a rate of 1.00, even though recording hasn't actually started yet.

Only one "pending record" condition can be pending for a particular Player at any one time. Immediately after a **OmPlrRecordDelay** function call the Player state changes from **omPlrCuerecord** to **omPlrRecord** – so most other functions are locked out. A **OmPlrRecord** command sent during the "pending record" interval will be quietly ignored; it will not start the recording. Sending a **OmPlrRecordAt** or a new **OmPlrRecordDelay** will update the "pending record" condition with the new information.

NOTE: OmPlrClose does not cancel a "pending record" condition. If the OmPlrDetach or OmPlrDetachAllClips functions can be used, they also do not cancel a "pending record" condition.

CAUTION: This function establishes a "pending record" condition. It is advisable to use a **OmPlrStop** function call as redundant insurance to remove any lingering "pending record" condition when starting a new sequence on the timeline. Note that if there are multiple connections to one Player, then any of the connections can send functions that will cause the "pending record" condition to be changed.

See also

Clip Function Discussion, *OmPlrCueRecord*, *OmPlrRecord*, *OmPlrRecordAt*

OmPlrSetPos()

Description

Sets the current position on the timeline.

```
OmPlrError OmPlrSetPos(
OmPlrHandle playerHandle,
int timelinePos);
```

Parameters ;

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
timelinePos	A numeric value to change the current position on the timeline.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Typically this function is used before play or recording begins to position the timeline at the desired starting point. The function can also be used while the timeline is stopped or while the timeline is playing. Changes of position while paused or in motion provide a "goto" functionality. The "**Implementation Hint**" below has some suggestions for handling a change of the current position while playing. Changing the position causes the Player to broadcast out media frames from the new position if the Player's timeline state is **omPlrCueplay** or **omPlrPlay**. An **OmPlrSetPos** command while the Player's timeline status is **omPlrRecord** while be ignored and a value of **omPlrWrongState** will be returned; this includes the period of "pending record" that is explained for the **OmPlrRecordAt** and the **OmPlrRecordDelay** commands.

Timeline position will typically be a positive value, but can also have negative values. During non-Looping operation, attempts to set the current position to a value less than **minPos** will be thwarted – the new value instead will be set to the same value as **minPos**. Similarly, attempts to set current position to a number greater or equal to **maxPos**, will result in a value of "**maxPos** –1."

This function also works during looping operation. If the function is being used to set a position that is outside the looping limits, a new position value will be calculated using modulo math that will be inside the looping limits. For instance if the loop limits were 300 to 400 and **OmPlrSetPos** was used to set a limit of 450, the new timeline position would be wrapped back to become 350 (= 450 – high limit + low limit).

Implementation Hint

Changing the position may cause temporary freeze frames. If the timeline is not in motion but will soon be in motion, use the function sequence of **OmPlrSetPos**, **OmPlrCuePlay** (at the desired speed), and **OmPlrPlay** (at the desired speed) to guarantee that there will not be any freeze frames on the transition into play. The call to **OmPlrCuePlay** will block until the Player has prefetched enough data to start playing cleanly.

However, if the timeline is already in motion and you need to use the **OmPlrSetPos** function, consider instead modifying the timeline contents to get the effect of a **setPos**.

For example:

If you have two clips attached — clipOne and clipTwo. You are currently playing clipOne and are somewhere in the middle of it. Now you want to "jump" to the beginning of clipTwo.

You should change the out point of clipOne to be about 2 seconds ahead of the current play position. In two seconds you will be in clipTwo.

That example was for the case of jumping ahead to the start of the next clip. If you needed to jump within the current clip, then you would need to change the out point of the current clip and attach a second copy of the current clip.

CAUTION: If setting both timeline position and changing the timeline's **minPos** or **maxPos** parameter, call this function after setting the other timeline position parameters. This avoids having the position altered by the setting of new values for **minPos** and **maxPos**

See also

OmPlrOpen, *Timeline Function Discussion*, *OmPlrSetMaxPos*, *OmPlrSetMinPos*, *OmPlrLoop*

OmPlrSetPosD

Description

Sets the current position on the timeline.

```
OmPlrError OmPlrSetPosD(
OmPlrHandle plrHandle,
double pos);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
double pos	A numeric value to change the current position on the timeline.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Similar to *OmPlrSetPos()* except takes position as a double floating point number instead of an int.

See also

OmPlrOpen, *OmPlrSetPos()*

OmPlrStep()

Description

Steps the timeline by a signed N frames.

```
OmPlrError OmPlrStep(
OmPlrHandle playerHandle,
int step);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
step	A signed numeric value, that gives the number of frames to be moved. Positive numbers cause stepping in the forward direction. The value of 00 is legal but not very exciting.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

The function returns an error of **omPlrWrongState** if the timeline state is not **omPlrPlay**.

Remarks

Only has an effect if the timeline state is already **omPlrPlay**. Forces the speed (the rate) to a value of 0.0. The stepping will not go outside the bounds of the Player's **minPos** and **maxPos** parameters. If the step parameter would take the position beyond the limit, the position instead is set to the limit; no error is returned. Step also functions during loop operation; if the step would have taken it beyond the loop limits then modulo math is applied to wrap the new position back around the other limit. For example, if the loop limits are 300 to 400 and the current position is 395, then a step of +10 will end up at position 305.

See also

OmPlrOpen, *OmPlrPlay*

OmPlrStepD

Description

Steps the timeline by a signed N frames.

```
OmPlrError OmPlrStep(
OmPlrHandle playerHandle,
double step);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
step	A signed numeric value, that gives the number of frames to be moved. Positive numbers cause stepping in the forward direction. A value of 00 is legal.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

The function returns an error of **omPlrWrongState** if the timeline state is not **omPlrPlay**.

Remarks

Only has an effect if the timeline state is already **omPlrPlay**. Forces the speed (the rate) to a value of 0.0. The stepping will not go outside the bounds of the Player's **minPos** and **maxPos** parameters. If the step parameter would take the position beyond the limit, the position instead is set to the limit; no error is returned. Step also functions during loop operation; if the step would have taken it beyond the loop limits then modulo math is applied to wrap the new position back around the other limit. For example, if the loop limits are 300 to 400 and the current position is 395, then a step of +10 will end up at position 305.

See also

OmPlrOpen, *OmPlrPlay*

OmPlrStop

Description

Stops the timeline motion.

```
OmPlrError OmPlrStop(
OmPlrHandle playerHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
double pos	A numeric value to change the current position on the timeline.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Stops any timeline motion, including all forms of record. The Player state becomes **omPlrStop**. The MediaDirector stops broadcasting media data onto the Spectrum 1394 network. In the typical case of using Spectrum MediaPort devices, the output signal will be connected to the input. Does not cause any changes on the timeline – all attached clips stay attached without any changes. Does not change the rate value that is reported as part of the Player status.

See also

OmPlrPlay, *OmPlrRecord*, *Timeline Function Discussion*

Chapter 3
Player Status Functions

This section provides Spectrum API functions that request the status of the Players. The following functions are included:

Function	Description
OmPlrGetPlayerStatus	Return status of Player for “this” connection.
OmPlrGetPlayerStatus1	Return status of Player for “this” connection.
OmPlrGetPlayerStatus2	Return status of Player for “this” connection.
OmPlrGetPlayerStatus3	Return status of Player for “this” connection.
OmPlrGetPos	Return the current position on the timeline.
OmPlrGetPosD	Return the current position on the timeline.
OmPlrGetPosAndClip	Query current timeline position; query for current timeline clip.
OmPlrGetPosInClip	Query clip relative position in clip at current timeline position.
OmPlrGetRecordTime	Query for available recording time.
OmPlrGetState	Return the state of the Player.
OmPlrGetTime	Query timeline position and system frame count; return raw data of ITC reference time. Also returns timecode from the timecode generator and from the video media.
OmPlrGetTime	Same as *OmPlrGetTime*. Also returns MediaDirector time of day in seconds and milliseconds.
OmPlrRegisterChangeCallback	Registers callback for player changes. It limits the need to poll with OmPlrGetPlayerStatus. Typical usage of this callback should be to simply post a semaphore or message to another thread that calls OmPlrGetPlayerStatus, etc. to determine the changes.

The following functions are alternatives to using *OmPlrGetPlayerStatus*. These functions query individual fields within the Player status. Refer to *OmPlrGetPlayerStatus* for a discussion of the status data fields.

Function	Description
OmPlrGetFrameRate	Get Player’s frame rate from its configuration.
OmPlrGetLoop	Get first and last timeline positions for playback looping mode.
OmPlrGetMaxPos	Return the maximum allowed position on timeline.
OmPlrGetMinPos	Return the minimum allowed position on timeline.
OmPlrGetNumClips	Return the number of clips attached to timeline.
OmPlrGetRate	Get timeline motion rate.
OmPlrPlayEnabled	Returns true if the Player is allowed to play back clips.
OmPlrRecordEnabled	Returns true if the Player is allowed to record clips.
OmPlrGetPosOfClip	Returns same **clipAttachPoint** data as **OmPlrGetClipData**.

OmPlrGetPlayerStatus

Description

Returns the status of the Player for "this" connection.

```
OmPlrError OmPlrGetPlayerStatus(
OmPlrHandleplayerHandle,
OmPlrStatus*pPlayerStatus);
```

NOTE: *OmPlrGetPlayerStatus* contains less status than *OmPlrGetPlayerStatus1* and does not work properly in Unicode applications where the name of the clip currently playing has a length of more than 31 characters. Unfortunately, *OmPlrGetPlayerStatus1* does not work with Spectrum software before version 4.6. Therefore, applications should first try calling *OmPlrGetPlayerStatus1*. If the call fails with an error omPlrNetworkBadVersion, then *OmPlrGetPlayerStatus* should be called.

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pPlayerStatus	A pointer to a **OmPlrStatus** structure that will be filled with the returned status. The buffer needs to have a length of at least "size of (**OmPlrStatus**)." Nothing will be written to the buffer if the function returns an error.
pPlayerStatus->size	This field within the **OmPlrStatus** structure must be initialized with the size of the structure before calling the **OmPlrGetPlayerStatus** function.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Definition of Fields in the OmPlrStatus Structure

The structure **OmPlrStatus** is defined in **omplrdefs.h**.
It is currently defined as:

```
struct OmPlrStatus {
uint    size;            // size of this structure, MUST
be initialzed
OmPlrState  state;       // Player state
int     pos;             // Player position
int     res1;
double  rate;            // Player rate
int     minPos;          // Player minimum position
int     maxPos;          // Player maximum position
uint    numClips;        // number of clips on the timeline
uint    clipListVersion; // version number of the timeline,
// increments on every timeline change
uint    currClipNum;     // current clip number (first clip on
```

```
// the timeline is 0)
int     currClipStartPos; // timeline pos of start of current
clip
uint    currClipIn;       // in point of current clip
uint    currClipOut;      // out point of current clip
uint    currClipLen;      // length (out - in) of current clip
uint    currClipFirstFrame;// first recorded frame of current
clip
uint    currClipLastFrame;// last recorded frame of current
clip
   char    currClipName[omPlrMaxClipNameLen];
// name of current clip
int     firstClipStartPos;// timeline pos of start of first
clip
int     lastClipEndPos;   // timeline pos of end of last clip
int     loopMin;          // minimum loop position
int     loopMax;          // maximum loop position
bool    playEnabled;      // true if Player is enabled for play
bool    recordEnabled;    // true if Player is enabled for
record
bool    resv2[2];
OmFrameRate frameRate;    // Player frame rate};
```

Remarks

These remarks expand on the code comments above in the structure definition.

state – defined by an enumeration in omplrdefs.h. Gives the current state of the Player's timeline.

pos – gives the current position of the Player's timeline.

rate – see *OmPlrPlay*.

minPos – in terms of timeline position.

maxPos – in terms of timeline position.

numClips – the number of clips on the timeline.

clipListVersion – a "marker" that changes whenever something effects the list of clips attached to the timeline. Increments if a clip is attached or deleted. Increments if the IN or OUT points of a clip are changed. Does not change if the timeline motion or rate are changed.

currClipNum – the first clip on the timeline is number 0. **currClipNum** is also 0 if no clips are attached. If no clips are attached (see **numClips**), all the rest of the curr…. fields are undefined garbage.

currClipStartPos – attachment point of current clip in terms of timeline frame numbering.

currClipIn – In point in terms of clip frame numbering. See *OmPlrAttach* and *OmPlrSetClipData*.

currClipOut – Out in terms of clip frame numbering.

currClipLen – always the same as the result of calculating **currClipOut** – **currClipIn**.

currClipFirstFrame – same information as found with *OmPlrClipGetInfo* for this clip.

currClipLastFrame – same information as found with *OmPlrClipGetInfo* for this clip

currClipName – the name of the clip currently playing.

firstClipStartPos – in terms of timeline frame numbering.

lastClipEndPos – in terms of timeline frame numbering.

loopMin – in terms of timeline frame numbering. **loopMin** is same as **loopMax** if looping is disabled. See *OmPlrLoop*.

loopMax – in terms of timeline frame numbering.

playEnabled – a boolean established as part of the Player configuration. Can't be changed by API functions.

recordEnabled – a boolean established as part of the Player configuration. Can't be changed by API functions. Player will be unable to record if this setting is false. A Player can be configured to be able to both record and playback.

frameRate – defined by an enumeration in omplrdefs.h. A Player can be configured to handle either NTSC or PAL frame rates but can't do a mix of both. Attempts to attach a clip of the wrong frame rate will fail and will return an error value.

- Do not get the NTSC frame rate mixed up with timecode modes; the NTSC frame rate is always **omFrmRate29_97Hz**. The current values are:

```
omFrmRateInvalid,
omFrmRate24Hz,
omFrmRate25Hz,
omFrmRate29_97Hz,
omFrmRate30Hz,
omFrmRate50Hz,
omFrmRate59_94Hz,
```

- A frame rate of **omFrmRate50Hz** or **omFrmRate59_94Hz** implies progressive rather than interlaced media; the units of the **pos** value will be 50Hz/second or 59Hz/second.

NOTE: Nothing is written to the buffer if an error is returned.

OmPlrGetPlayerStatus1

Description

Returns the status of the Player for "this" connection. Use instead of the original *OmPlrGetPlayerStatus*.

```
OmPlrError OmPlrGetPlayerStatus1(
OmPlrHandle playerHandle,
OmPlrStatus1 *pPlayerStatus);
```

NOTE: *OmPlrGetPlayerStatus* contains less status than *OmPlrGetPlayerStatus1* and does not work properly in Unicode applications where the name of the clip currently playing has a length of more than 31 characters. Unfortunately, *OmPlrGetPlayerStatus1* does not work with Spectrum software before version 4.6. Therefore, applications should first try calling *OmPlrGetPlayerStatus1*. If the call fails with an error omPlrNetworkBadVersion, then call *OmPlrGetPlayerStatus*.

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pPlayerStatus	A pointer to a **OmPlrStatus** structure that will be filled with the returned status. The buffer needs to have a length of at least "size of (**OmPlrStatus**)." Nothing will be written to the buffer if the function returns an error.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Definition of Fields in the OmPlrStatus Structure

The structure **OmPlrStatus** is defined in **omplrdefs.h**.
It is currently defined as:

```
struct OmPlrStatus1 {
uint    version;            // internal use only
OmPlrState  state;        // Player state
double rate;              // Player rate
int     pos;              // Player position
int     minPos;           // Player minimum position
int     maxPos;           // Player maximum position
uint    numClips;         // number of clips on the timeline
uint    clipListVersion;  // version number of the timeline,
// increments on every timeline change
OmPlrClipHandle currClipHandle; //handle to current clip
uint    currClipNum;      // current clip number (first clip on
// the timeline is 0)
int     currClipStartPos; // timeline pos of start of current
clip
uint    currClipIn;       // in point of current clip
uint    currClipOut;      // out point of current clip
uint    currClipLen;      // length (out - in) of current clip
uint    currClipFirstFrame;// first recorded frame of current
clip
uint    currClipLastFrame;// last recorded frame of current
clip
OmFrameRate currClipFrameRate; //frame rate of current clip
   OmTcData currClipStartTimecode; //start timecode of current
clip
char    currClipName[omPlrMaxClipNameLen];
// name of current clip
int     firstClipStartPos;// timeline pos of start of first
clip
int     lastClipEndPos;   // timeline pos of end of last clip
int     loopMin;          // minimum loop position
int     loopMax;          // maximum loop position
```

```
        bool    playEnabled;      // true if Player is enabled for play
        bool    recordEnabled;    // true if Player is enabled for
        record
        bool    dropFrame; // true if Player set to drop frame mode
        bool    tcgPlayInsert; // true if tcg inserting on play
        bool    tcgRecordInsert; // true if tcg inserting on record
        OmPlrTcgMode tcgMode; //tcg mode
        OmFrameRate frameRate;    // Player frame rate
};
```

Remarks

These remarks expand on the code comments above in the structure definition.

NOTE: Nothing is written to the buffer if an error is returned.

state – defined by an enumeration in omplrdefs.h. Gives the current state of the Player's timeline.

pos – gives the current position of the Player's timeline.

rate – see *OmPlrPlay*.

minPos – in terms of timeline position.

maxPos – in terms of timeline position.

numClips – the number of clips on the timeline.

clipListVersion – a "marker" that changes whenever something effects the list of clips attached to the timeline. Increments if a clip is attached or deleted. Increments if the IN or OUT points of a clip are changed. Does not change if the timeline motion or rate are changed.

currClipHandle - handle to current clip..

currClipNum – the first clip on the timeline is number 0. **currClipNum** is also 0 if no clips are attached. If no clips are attached (see **numClips**), all the rest of the curr…. fields are undefined garbage.

currClipStartPos – attachment point of current clip in terms of timeline frame numbering.

currClipIn – In point in terms of clip frame numbering. See *OmPlrAttach* and *OmPlrSetClipData*.

currClipOut – Out in terms of clip frame numbering.

currClipLen – always the same as the result of calculating **currClipOut** – **currClipIn**.

currClipFirstFrame – same information as found with *OmPlrClipGetInfo* for this clip.

currClipLastFrame – same information as found with *OmPlrClipGetInfo* for this clip

currClipFrameRate – frame rate of the currently playing clip.

currClipStartTimecode – start timecode of the currently playing clip.

currClipName – the name of the clip currently playing.

firstClipStartPos – in terms of timeline frame numbering.

lastClipEndPos – in terms of timeline frame numbering.

loopMin – in terms of timeline frame numbering. **loopMin** is same as **loopMax** if looping is disabled. See *OmPlrLoop*.

loopMax – in terms of timeline frame numbering.

playEnabled – a boolean established as part of the Player configuration. Can't be changed by API functions.

recordEnabled – a boolean established as part of the Player configuration. Can't be changed by API functions. Player will be unable to record if this setting is false. A Player can be configured to be able to both record and playback.

dropFrame – drop frame setting of the Player.

tcgPlayInsert – timecode generator playout insertion setting.

tcgRecordInsert – timecode generator record insertion setting.

tcgMode – timecode generator counter mode.

frameRate – defined by an enumeration in omplrdefs.h. A Player can be configured to handle either NTSC or PAL frame rates but can't do a mix of both. Attempts to attach a clip of the wrong frame rate will fail and will return an error value.

- Do not get the NTSC frame rate mixed up with timecode modes; the NTSC frame rate is always **omFrmRate29_97Hz**. The current values are:

```
omFrmRateInvalid,
omFrmRate24Hz,
omFrmRate25Hz,
omFrmRate29_97Hz,
omFrmRate30Hz,
omFrmRate50Hz,
omFrmRate59_94Hz,
```

- A frame rate of **omFrmRate50Hz** or **omFrmRate59_94Hz** implies progressive rather than interlaced media; the units of the **pos** value will be 50Hz/second or 59Hz/second.

OmPlrGetPlayerStatus2

Description

Returns the status of the Player for "this" connection. Use instead of the original *OmPlrGetPlayerStatus*.

```
OmPlrError OmPlrGetPlayerStatus2(
OmPlrHandle playerHandle,
OmPlrStatus2 *pPlayerStatus);
```

NOTE: *OmPlrGetPlayerStatus2* does not work with Spectrum software before version 4.7 SR3; applications should first try calling *OmPlrGetPlayerStatus2*. If the call fails with an error omPlrNetworkBadVersion, then call *OmPlrGetPlayerStatus1*.

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pPlayerStatus	A pointer to a **OmPlrStatus2** structure that will be filled with the returned status. The buffer needs to have a length of at least "size of (**OmPlrStatus2**)." Nothing will be written to the buffer if the function returns an error.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Definition of Fields in the OmPlrStatus2 Structure

The structure **OmPlrStatus2** is defined in **omplrdefs.h**. It is currently defined as:

```
struct OmPlrStatus2 {
uint    version;            // internal use only
OmPlrState  state;        // Player state
double rate;              // Player rate
int     pos;              // Player position
int     minPos;           // Player minimum position
int     maxPos;           // Player maximum position
uint    numClips;         // number of clips on the timeline
uint    clipListVersion;  // version number of the timeline,
// increments on every timeline change
OmPlrClipHandle currClipHandle; //handle to current clip
uint    currClipNum;      // current clip number (first clip on
// the timeline is 0)
int     currClipStartPos; // timeline pos of start of current
clip
uint    currClipIn;       // in point of current clip
uint    currClipOut;      // out point of current clip
uint    currClipLen;      // length (out - in) of current clip
uint    currClipFirstFrame;// first recorded frame of current
clip
uint    currClipLastFrame;// last recorded frame of current
clip
OmFrameRate currClipFrameRate; //frame rate of current clip
   OmTcData currClipStartTimecode; //start timecode of current
clip
char    currClipName[omPlrMaxClipNameLen];
// name of current clip
int     firstClipStartPos;// timeline pos of start of first
clip
int     lastClipEndPos;   // timeline pos of end of last clip
int     loopMin;          // minimum loop position
int     loopMax;          // maximum loop position
```

```
bool    playEnabled;      // true if Player is enabled for play
bool    recordEnabled;    // true if Player is enabled for
record
bool    dropFrame; // true if Player set to drop frame mode
bool    tcgPlayInsert; // true if tcg inserting on play
bool    tcgRecordInsert; // true if tcg inserting on record
OmPlrTcgMode tcgMode; //tcg mode
OmFrameRate frameRate;    // Player frame rate
bool portDown: //true when I/O port is down
};
```

Remarks

These remarks expand on the code comments above in the structure definition.

NOTE: Nothing is written to the buffer if an error is returned.

state – defined by an enumeration in omplrdefs.h. Gives the current state of the Player's timeline.

pos – gives the current position of the Player's timeline.

rate – see *OmPlrPlay*.

minPos – in terms of timeline position.

maxPos – in terms of timeline position.

numClips – the number of clips on the timeline.

clipListVersion – a "marker" that changes whenever something effects the list of clips attached to the timeline. Increments if a clip is attached or deleted. Increments if the IN or OUT points of a clip are changed. Does not change if the timeline motion or rate are changed.

currClipHandle - handle to current clip.

currClipNum – the first clip on the timeline is number 0. **currClipNum** is also 0 if no clips are attached. If no clips are attached (see **numClips**), all the rest of the curr…. fields are undefined garbage.

currClipStartPos – attachment point of current clip in terms of timeline frame numbering.

currClipIn – In point in terms of clip frame numbering. See *OmPlrAttach* and *OmPlrSetClipData*.

currClipOut – Out in terms of clip frame numbering.

currClipLen – always the same as the result of calculating **currClipOut** – **currClipIn**.

currClipFirstFrame – same information as found with *OmPlrClipGetInfo* for this clip.

currClipLastFrame – same information as found with *OmPlrClipGetInfo* for this clip

currClipFrameRate – frame rate of the currently playing clip.

currClipStartTimecode – start timecode of the currently playing clip.

currClipName – the name of the clip currently playing.

firstClipStartPos – in terms of timeline frame numbering.

lastClipEndPos – in terms of timeline frame numbering.

loopMin – in terms of timeline frame numbering. **loopMin** is same as **loopMax** if looping is disabled. See *OmPlrLoop*.

loopMax – in terms of timeline frame numbering.

playEnabled – a boolean established as part of the Player configuration. This cannot be changed by API functions.

recordEnabled – a boolean established as part of the Player configuration. Can't be changed by API functions. Player will be unable to record if this setting is false. A Player can be configured to be able to both record and playback.

dropFrame – drop frame setting of the Player.

tcgPlayInsert – timecode generator playout insertion setting.

tcgRecordInsert – timecode generator record insertion setting.

tcgMode – timecode generator counter mode.

frameRate – defined by an enumeration in omplrdefs.h. A Player can be configured to handle either NTSC or PAL frame rates but can't do a mix of both. Attempts to attach a clip of the wrong frame rate will fail and will return an error value.

- Do not get the NTSC frame rate mixed up with timecode modes; the NTSC frame rate is always **omFrmRate29_97Hz**. The current values are:

```
omFrmRateInvalid,
omFrmRate24Hz,
omFrmRate25Hz,
omFrmRate29_97Hz,
omFrmRate30Hz,
omFrmRate50Hz,
omFrmRate59_94Hz,
```

- A frame rate of **omFrmRate50Hz** or **omFrmRate59_94Hz** implies progressive rather than interlaced media; the units of the **pos** value will be 50Hz/second or 59Hz/second.

portDown – a boolean that indicates the MediaPort is down, disconnected, powered off, or otherwise inaccessible.

OmPlrGetPlayerStatus3

Description

Returns the status of the Player for "this" connection. Use instead of the original *OmPlrGetPlayerStatus*.

```
OmPlrError OmPlrGetPlayerStatus3(
OmPlrHandle playerHandle,
OmPlrStatus3 *pPlayerStatus);
```

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pPlayerStatus	A pointer to a **OmPlrStatus3** structure that will be filled with the returned status. The buffer needs to have a length of at least "size of (**OmPlrStatus3**)." Nothing will be written to the buffer if the function returns an error.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Definition of Fields in the OmPlrStatus3 Structure

The structure **OmPlrStatus3** is defined in **omplrdefs.h**. It is currently defined as:

```
struct OmPlrStatus3A {
uint    version;             // internal use only
OmPlrState  state;        // Player state
double rate;              // Player rate
int     pos;              // Player position
int     minPos;           // Player minimum position
int     maxPos;           // Player maximum position
uint    numClips;         // number of clips on the timeline
uint    clipListVersion;  // version number of the timeline,
// increments on every timeline change
OmPlrClipHandle currClipHandle; //handle to current clip
uint    currClipNum;      // current clip number (first clip on
// the timeline is 0)
int     currClipStartPos; // timeline pos of start of current
clip
uint    currClipIn;       // in point of current clip
uint    currClipOut;      // out point of current clip
uint    currClipLen;      // length (out - in) of current clip
uint    currClipFirstFrame;// first recorded frame of current
clip
uint    currClipLastFrame;// last recorded frame of current
clip
OmFrameRate currClipFrameRate; //frame rate of current clip
   OmTcData currClipStartTimecode; //start timecode of current
clip
char    currClipName[omPlrMaxClipNameLen];
// name of current clip
int     firstClipStartPos;// timeline pos of start of first
clip
int     lastClipEndPos;   // timeline pos of end of last clip
int     loopMin;          // minimum loop position
int     loopMax;          // maximum loop position
bool    playEnabled;      // true if Player is enabled for play
bool    recordEnabled;    // true if Player is enabled for
record
bool    dropFrame; // true if Player set to drop frame mode
bool    tcgPlayInsert; // true if tcg inserting on play
bool    tcgRecordInsert; // true if tcg inserting on record
OmPlrTcgMode tcgMode; //tcg mode
OmFrameRate frameRate;    // Player frame rate
bool portDown: //true when I/O port is down
bool refLocked; //true when locked to reference input
uint recBlackCount; //number of black video frames recorded,
reset to zero on cueRecord
```

```
};
```

Remarks

These remarks expand on the code comments above in the structure definition.

NOTE: Nothing is written to the buffer if an error is returned.

state – defined by an enumeration in omplrdefs.h. Gives the current state of the Player's timeline.

pos – gives the current position of the Player's timeline.

rate – see *OmPlrPlay*.

minPos – in terms of timeline position.

maxPos – in terms of timeline position.

numClips – the number of clips on the timeline.

clipListVersion – a "marker" that changes whenever something effects the list of clips attached to the timeline. Increments if a clip is attached or deleted. Increments if the IN or OUT points of a clip are changed. Does not change if the timeline motion or rate are changed.

currClipHandle - handle to current clip.

currClipNum – the first clip on the timeline is number 0. **currClipNum** is also 0 if no clips are attached. If no clips are attached (see **numClips**), all the rest of the curr…. fields are undefined garbage.

currClipStartPos – attachment point of current clip in terms of timeline frame numbering.

currClipIn – In point in terms of clip frame numbering. See *OmPlrAttach* and *OmPlrSetClipData*.

currClipOut – Out in terms of clip frame numbering.

currClipLen – always the same as the result of calculating **currClipOut** – **currClipIn**.

currClipFirstFrame – same information as found with *OmPlrClipGetInfo* for this clip.

currClipLastFrame – same information as found with *OmPlrClipGetInfo* for this clip

currClipFrameRate – frame rate of the currently playing clip.

currClipStartTimecode – start timecode of the currently playing clip.

currClipName – the name of the clip currently playing.

firstClipStartPos – in terms of timeline frame numbering.

lastClipEndPos – in terms of timeline frame numbering.

loopMin – in terms of timeline frame numbering. **loopMin** is same as **loopMax** if looping is disabled. See *OmPlrLoop*.

loopMax – in terms of timeline frame numbering.

playEnabled – a boolean established as part of the Player configuration. This cannot be changed by API functions.

recordEnabled – a boolean established as part of the Player configuration. Can't be changed by API functions. Player will be unable to record if this setting is false. A Player can be configured to be able to both record and playback.

dropFrame – drop frame setting of the Player.

tcgPlayInsert – timecode generator playout insertion setting.

tcgRecordInsert – timecode generator record insertion setting.

tcgMode – timecode generator counter mode.

frameRate – defined by an enumeration in omplrdefs.h. A Player can be configured to handle either NTSC or PAL frame rates but can't do a mix of both. Attempts to attach a clip of the wrong frame rate will fail and will return an error value.

- Do not get the NTSC frame rate mixed up with timecode modes; the NTSC frame rate is always **omFrmRate29_97Hz**. The current values are:

```
omFrmRateInvalid,
omFrmRate24Hz,
omFrmRate25Hz,
omFrmRate29_97Hz,
omFrmRate30Hz,
omFrmRate50Hz,
omFrmRate59_94Hz,
```

- A frame rate of **omFrmRate50Hz** or **omFrmRate59_94Hz** implies progressive rather than interlaced media; the units of the **pos** value will be 50Hz/second or 59Hz/second.

portDown – a boolean that indicates the MediaPort is down, disconnected, powered off, or otherwise inaccessible.

refLocked – reference is locked to the input

recBlackCount – number of black video frames recorded, reset to zero on cueRecord

OmPlrGetPos

Description

Returns the current position on the timeline.

```
OmPlrError OmPlrGetPos(
OmPlrHandle playerHandle,
int *pTimelinePos);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pTimelinePos	A pointer that is used to return the current position on the timeline. If the function returns an error, nothing will be written using the pointer.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Timeline position will typically be a positive value, but can also have negative values. For instance, if a negative value was used as an argument for **OmPlrSetMinPos** and **OmPlrSetPos**.

This same information is available from the **OmPlrGetPlayerStatus** function.

See also

Timeline Function Discussion, *OmPlrSetPos()*, *OmPlrGetPlayerStatus*

OmPlrGetPosD

Description

Returns the current position on the timeline.

```
OmPlrError OmPlrGetPosD(
OmPlrHandle plrHandle,
double *pPos);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pPos	A pointer that is used to return the current position on the timeline. If the function returns an error, nothing will be written using the pointer.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Similar to *OmPlrGetPos* except that it returns a double instead of an int.

Timeline position will typically be a positive value, but can also have negative values. For instance, if a negative value was used as an argument for **OmPlrSetMinPos** and **OmPlrSetPos**.

See also

Timeline Function Discussion, *OmPlrSetPos()*, *OmPlrGetPlayerStatus*

OmPlrGetPosAndClip

Description

Queries current timeline position; also query about which is the current clip being played on the timeline.

```
OmPlrError OmPlrGetPosAndClip(
OmPlrHandle playerHandle,
```

```
int *pTimelinePosition,
OmPlrClipHandle *pOmPlrClipHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pTimelinePosition	A pointer that is used to return the current position on the timeline. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.
pOmPlrClipHandle	A pointer that is used to return the **OmPlrClipHandle** for the current clip on the timeline. This **OmPlrClipHandle** is the same handle value that was returned by the function **OmPlrAttach**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

NOTE: A valid OmPlrClipHandle will never have a value of 00.

The timeline position information is available from the **OmPlrGetPlayerStatus** function. The handle for the current clip can also be obtained using the **currNum** field of the **OmPlrStatus** structure and the function **OmPlrGetClipAtNum**.

See also

OmPlrOpen, *OmPlrGetPlayerStatus*, *OmPlrGetClipAtNum*, *OmPlrAttach*

OmPlrGetPosInClip

Description

Queries clip relative position in the clip at the current timeline position.

```
OmPlrError OmPlrGetPosInClip(
OmPlrHandle playerHandle,
uint *pClipPosition);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipPosition	A pointer used to return the clip relative position within the current clip on the timeline. If the function returns an error, nothing will be written using the pointer.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The position written back is relative to the clip. The position is absolute within the clip. For instance, if the clip was attached with an IN point of 34, then this function would return 34 when the timeline is positioned at that IN frame.

If there is no clip attached at the current timeline position, then a value of 00 is written back to **ClipPosition**.

This function does not identify the current clip. The values within the data returned by **OmPlrGetPlayerStatus** will allow you to calculate the position within the current clip and at the same time will give you the name of the current clip.

See also

OmPlrOpen

OmPlrGetRecordTime

Description

Queries for the available recording time, assuming the configuration of "this" Player object.

```
OmPlrError OmPlrGetRecordTime(
OmPlrHandle playerHandle,
uint *pSeconds);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pSeconds	A required pointer that is used to write back a binary number that gives the available recording time in units of seconds.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The available space will be some number of bytes. The conversion of bytes to time assumes the recording rate of the Player object for this Player handle. The number returned by this function only depends on the configuration. It will not change depending on the recording state of the Player. This function works even if the Player is configured for playback only. The number should be used carefully – more or less than this amount of time will be available for recording for this Player depending on the activity of other players and other users of the file system. And the recording time will be an estimate.

The function **OmPlrClipGetFsSpace** returns the space available in units of bytes free and bytes used.

See also

OmPlrOpen, *OmPlrClipGetFsSpace*

OmPlrGetState

Description

Returns the state of the Player.

```
OmPlrError OmPlrGetState(
OmPlrHandle playerHandle,
OmPlrState *pPlayerState);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pPlayerState	A pointer used to return an **OmPlrState** value that gives the state of the Player. Nothing is returned via the pointer if the function returns an error condition. **OmPlrState** is an enumeration so its size is the same as int.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The definition of **OmPlrState** is an enumeration defined in the file omplrdefs.h. The current values of **OmPlrState** are:

```
enum OmPlrState {
omPlrStopped,
omPlrCuePlay,
omPlrPlay,
omPlrCuerecord,
omPlrRecord
};
```

This same information is available from the **OmPlrGetPlayerStatus** function.

See also

OmPlrOpen, *OmPlrPlay*, *OmPlrStop*, *OmPlrCueRecord*, *OmPlrCuePlay*, *OmPlrRecord*

OmPlrGetTime

Description

Queries timeline position and system frame count; return raw data of VITC reference time. Also returns timecode from the timecode generator and video media timecode.

```
OmPlrError OmPlrGetTime(
OmPlrHandle playerHandle,
int *pTimelinePosition,
uint *pSysFrame,
uint *pSysFrameFrac,
OmTcData *pRefVitc,
OmTcData *pTcGen,
OmTcData *pVideoTime);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pTimelinePosition	A pointer for returning the current position on the timeline. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.
pSysFrame	A pointer for returning the integral part of the current value of the MediaDirector's system frame counter. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.
pSysFrameFrac	A pointer for returning the fractional part of the current value of the MediaDirector's system frame counter. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.

Parameter	Description
pRefVitc	A pointer that is used to return the most recently read VITC time that was extracted from the MediaDirector's analog reference signal. The buffer pointed at needs to be at least as large as the **OmTcData** structure. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer. The values written will be all 0xFF if no valid VITC is available on the analog reference.
pTcGen	A pointer that is used to return the current time and user bits from the MediaDirector's internal timecode generator. The buffer pointed at needs to be at least as large as the **OmTcData** structure. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer. The values written will be all 0xFF if no valid timecode can be obtained. See the *Timecode Discussion* for more information.
pVideoTime	A pointer that is used to return the most recently read time + user bits that was read by the MediaDirector's internal timecode reader. The buffer pointed at needs to be at least as large as the **OmTcData** structure. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer. The values written will be all 0xFF if no valid timecode is available. See the *Timecode Discussion* for more information.

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This function is an important tool for relating the various time objects inside a Player. This function provides a snapshot of a system frame counter along with snapshots of the other times that were taken at the same moment. You can use the values returned by this function to establish an offset between these values. The function returns a snapshot of the current position of the timeline so that the returned values can be used to establish the offset between the timeline position and the system frame counter.

There is a system frame counter for each player that continuously counts at the player frame rate with a clock derived from reference. This counter has enough bits to count out to about 4.5 years at 30Hz. It is cleared to 0 whenever the MediaDirector is reset. It is not possible to set the system frame counter. You can discover the frame rate of the system frame counter by using the function *OmPlrGetFrameRate*. The **SysFrameFrac** is the fractional part of the system frame counter. It is a 32 bit number that represents the value (**SysFrameFrac**)/(2^32); for instance a **SysFrameFrac** value of 0x8000 0000 represents ½. Another way of expressing the same relationship is:

double **frameCount** = **sysFrame** + ((double)(**sysFrameFrac**)) / (0x1 0000 0000 L)

The **RefVitc** data is the raw data straight from the MediaDirector's VITC reader hardware. The VITC signal is extracted from the analog reference that is connected to the MediaDirector. The data is updated once per field. The reading will be from "this" field and does not need any adjusting for processing delays. Part of the MediaDirector setup is to specify which video lines contain the VITC; this information is needed by the VITC reading hardware. The VITC lines setting is part of the configuration process within the Spectrum SystemManager product. The

VITC signal will normally be a house time that is incrementing steadily at a 1X rate. The MediaDirector does not directly make use of the VITC signal; it only reads the signal and makes it available with this function so that the application can then execute timed functions such as **OmPlrPlayAt**. The MediaDirector can operate even if VITC is not present.

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.

The structure looks like:

```
struct OmTcData {
// Byte 0:
// Frame number (units):
uint framesUnits:4;
// Binary group 1:
uint binaryGroup1:4;

// Byte 1:
// Frame number (tens):
uint framesTens:2;
   // TRUE ==> drop-frame, FALSE ==> non-drop-frame, for
FldRate59_94Hz;
// reserved, for FldRate50Hz:
uint dropFrame_59_94:1;
// Color-frame marker:
uint colorFrame:1;
// Binary group 2:
uint binaryGroup2:4;

// Byte 2:
uint secondsUnits:4;
// Binary group 3:
uint binaryGroup3:4;
// Byte 3:
// Seconds (tens):
uint secondsTens:3;
   // NRZ phase correction for LTC FldRate59_94Hz;
   // Binary group flag #0 for LTC/VITC FldRate50Hz:
// Fieldmark for VITC FldRate59_94Hz
uint pcFm_59_94_bg0_50:1;
// Binary group 4:
uint binaryGroup4:4;

// Byte 4:
// Minutes (units):
uint minutesUnits:4;
// Binary group 5:
uint binaryGroup5:4;

// Byte 5:
```

```
            // Minutes (tens):
            uint minutesTens:3;
               // Binary group flag #0 for FldRate59_94Hz;
               // binary group flag #2 for FldRate50Hz:
            uint bg0_59_94_bg2_50:1;
            // Binary group 6:
            uint binaryGroup6:4;

            // Byte 6:
            // Hours (units):
            uint hoursUnits:4;
            // Binary group 7:
            uint binaryGroup7:4;

            // Byte 7:
            // Hours (tens):
            uint hoursTens:2;
               // Binary group flag #1 (for both FldRate59_94Hz and
            FldRate50Hz):
            uint bg1:1;
            // Binary group flag #2 for FldRate59_94Hz;
               // NRZ phase correction for LTC FldRate50Hz:
            // Fieldmark for VITC FldRate50Hz:
            uint bg2_59_94_pcFm_50:1;
            // Binary group 8:
            uint binaryGroup8:4;
};
```

This function will work even if the connection was opened with a NULL Player. In the case of a NULL Player the position information will be 00 and the **tcGen** value will be invalid and the videoTime values will be invalid.

The **pTcGen** and the **pVideoTime** pointers only return valid times if the connection was opened with a non-NULL player. In addition, the videoTime will not be valid if the player device is in the STOP mode. When the times are not valid, the **OmTcData** structure is filled with values of 0xFF.

See also

OmPlrOpen, *OmPlrGetFrameRate*, *OmPlrPlayAt*, *OmPlrRecordAt*, *Timecode Discussion*

OmPlrGetTime1

Description

Same as *OmPlrGetTime*. Also returns MediaDirector time of day in seconds and milliseconds.

```
OmPlrError OmPlrGetTime1(
OmPlrHandle plrHandle,
int *pPosition,
uint *pSysFrame = 0,
uint *pSysFrameFrac = 0,
OmTcData *pRefVitc = 0,
OmTcData *pTcg = 0,
OmTcData *pVideoTc = 0,
```

```
            uint *seconds = 0,
            unit *milliseconds = 0);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pTimelinePosition	A pointer for returning the current position on the timeline. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.
pSysFrame	A pointer for returning the integral part of the current value of the MediaDirector's system frame counter. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.
pSysFrameFrac	A pointer for returning the fractional part of the current value of the MediaDirector's system frame counter. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.
pRefVitc	A pointer that is used to return the most recently read VITC time that was extracted from the MediaDirector's analog reference signal. The buffer pointed at needs to be at least as large as the **OmTcData** structure. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer. The values written will be all 0xFF if no valid VITC is available on the analog reference.
pTcg	A pointer that is used to return the current time and user bits from the MediaDirector's internal timecode generator. The buffer pointed at needs to be at least as large as the **OmTcData** structure. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer. The values written will be all 0xFF if no valid timecode can be obtained. See the *Timecode Discussion* for more information.
pVideoTc	A pointer that is used to return the most recently read time + user bits that was read by the MediaDirector's internal timecode reader. The buffer pointed at needs to be at least as large as the **OmTcData** structure. If you do not want to have this value returned, then pass 00 f or the pointer. If the function returns an error, nothing will be written using the pointer. The values written will be all 0xFF if no valid timecode is available. See the *Timecode Discussion* for more information.
***seconds**	Returns MediaDirector time of day in seconds.
***milliseconds**	Returns MediaDirector time of day in milliseconds.

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Similar to *OmPlrGetTime* but returns more data.

This function is an important tool for relating the various time objects inside a Player. This function provides a snapshot of a system frame counter along with snapshots of the other times that were taken at the same moment. You can use the values returned by this function to establish an offset between these values. The function returns a snapshot of the current position of the timeline so that the returned values can be used to establish the offset between the timeline position and the system frame counter.

There is a system frame counter for each player that continuously counts at the player frame rate with a clock derived from reference. This counter has enough bits to count out to about 4.5 years at 30Hz. It is cleared to 0 whenever the MediaDirector is reset. It is not possible to set the system frame counter. You can discover the frame rate of the system frame counter by using the function *OmPlrGetFrameRate*. The **SysFrameFrac** is the fractional part of the system frame counter. It is a 32 bit number that represents the value (**SysFrameFrac**)/(2^32); for instance a **SysFrameFrac** value of 0x8000 0000 represents ½. Another way of expressing the same relationship is:

double **frameCount** = **sysFrame** + ([double][**sysFrameFrac**]) / (0x1 0000 0000 L)

The **RefVitc** data is the raw data straight from the MediaDirector's VITC reader hardware. The VITC signal is extracted from the analog reference that is connected to the MediaDirector. The data is updated once per field. The reading will be from "this" field and does not need any adjusting for processing delays. Part of the MediaDirector setup is to specify which video lines contain the VITC; this information is needed by the VITC reading hardware. The VITC lines setting is part of the configuration process within the Spectrum SystemManager product. The VITC signal will normally be a house time that is incrementing steadily at a 1X rate. The MediaDirector does not directly make use of the VITC signal; it only reads the signal and makes it available with this function so that the application can then execute timed functions such as **OmPlrPlayAt**. The MediaDirector can operate even if VITC is not present.

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. The structure looks like:

```
struct OmTcData {
// Byte 0:
// Frame number (units):
uint framesUnits:4;
// Binary group 1:
uint binaryGroup1:4;

// Byte 1:
// Frame number (tens):
uint framesTens:2;
   // TRUE ==> drop-frame, FALSE ==> non-drop-frame, for
FldRate59_94Hz;
// reserved, for FldRate50Hz:
uint dropFrame_59_94:1;
// Color-frame marker:
uint colorFrame:1;
// Binary group 2:
```

```
            uint binaryGroup2:4;

            // Byte 2:
            uint secondsUnits:4;
            // Binary group 3:
            uint binaryGroup3:4;

            // Byte 3:
            // Seconds (tens):
            uint secondsTens:3;
               // NRZ phase correction for LTC FldRate59_94Hz;
               // Binary group flag #0 for LTC/VITC FldRate50Hz:
            // Fieldmark for VITC FldRate59_94Hz
            uint pcFm_59_94_bg0_50:1;
            // Binary group 4:
            uint binaryGroup4:4;

            // Byte 4:
            // Minutes (units):
            uint minutesUnits:4;
            // Binary group 5:
            uint binaryGroup5:4;

            // Byte 5:
            // Minutes (tens):
            uint minutesTens:3;
               // Binary group flag #0 for FldRate59_94Hz;
               // binary group flag #2 for FldRate50Hz:
            uint bg0_59_94_bg2_50:1;
            // Binary group 6:
            uint binaryGroup6:4;

            // Byte 6:
            // Hours (units):
            uint hoursUnits:4;
            // Binary group 7:
            uint binaryGroup7:4;

            // Byte 7:
            // Hours (tens):
            uint hoursTens:2;
               // Binary group flag #1 (for both FldRate59_94Hz and
            FldRate50Hz):
            uint bg1:1;
               // Binary group flag #2 for FldRate59_94Hz;
               // NRZ phase correction for LTC FldRate50Hz:
            // Fieldmark for VITC FldRate50Hz:
            uint bg2_59_94_pcFm_50:1;
            // Binary group 8:
            uint binaryGroup8:4;
};
```

This function will work even if the connection was opened with a NULL Player. In the case of a NULL Player the position information will be 00 and the **tcGen** value will be invalid and the **videoTime** values will be invalid.

The **pTcg** and the **pVideoTc** pointers only return valid times if the connection was opened with a non-NULL player. In addition, the **videoTime** will not be valid if the player device is in the STOP mode. When the times are not valid, the **OmTcData** structure is filled with values of 0xFF.

Note that the seconds and milliseconds pointers return the MediaDirector time of day in seconds past January 1, 1970 GMT.

See also

OmPlrOpen, *OmPlrGetFrameRate*, *OmPlrPlayAt*, *OmPlrRecordAt*, *Timecode Discussion*

OmPlrRegisterChangeCallback

Description

Registers callback for player changes. It limits the need to poll with **OmPlrGetPlayerStatus**. Typical usage of this callback should be to simply post a semaphore or message to another thread that calls OmPlrGetPlayerStatus, etc. to determine the changes.

```
OmPlrError OmPlrRegisterChangeCallback(
OmPlrHandle plrHandle,
void (*pPlrChangedCallback)(void *param),
void *param);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pPlayerChangedCallback	A pointer to a function that will be called whenever player state or player timeline changes. The argument passed to the function will be the value of param. Use a value of 0 for pPlayerChangedCallback if you want to disable the callback.
Param	A numeric value that will be supplied to the pPlrChangedCallback function whenever it is called.

Return value

A numeric value that defines success or error. A value of 0 indicates “all ok.” Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The set of player changes that trigger this callback include:

OmPlrStatus.state – player state change

OmPlrStatus.rate – player rate change, limited to changes to or from 1.0

OmPlrStatus.minPos – minimum timeline position change

OmPlrStatus.maxPos – maximum timeline position change

OmPlrStatus.loopMin – timeline looping minimum position change

OmPlrStatus.loopMax – timeline looping maximum position change

OmPlrStatus.playEnabled – enabled for play change

OmPlrStatus.recordEnabled – enabled for record change

OmPlrStatus.portDown– port down change

OmPlrStatus.recBlackCount – black frames recorded

clip attached to the timeline

clip detached from the timeline

change of in-point or out-point of a clip on the timeline

See also

OmPlrClipRegisterCallbacks

Chapter 4
Player Timeline Functions

This section provides Spectrum API Functions related to the manipulation of the Timeline. The following functions are included.

Function	Description
OmPlrAttach	Used to attach a clip to the timeline, and create a new clip.
OmPlrAttach1	Version of *OmPlrAttach* that adds a modified track match string.
OmPlrAttach2	Version of *OmPlrAttach* that adds video format conversion options.
OmPlrAttach3	Version of *OmPlrAttach* that adds an AFD and secondary audio mix select.
OmPlrAttach4	Version of *OmPlrAttach* that adds the list of caption or subtitle files to insert during playback.
OmPlrDetach	Remove an individual clip from the timeline.
OmPlrDetachAllClips	Remove all clips from the timeline.
OmPlrGetClipAtNum	Used to obtain a handle for a clip on the timeline.
OmPlrGetClipAtPos	Used to obtain a handle for a clip on the timeline.
OmPlrGetClipData	Used to return information about a clip as it has been mounted on the timeline.
OmPlrGetClipData1	Used to return information about a clip on the timeline.
OmPlrGetClipName	Get the name of a clip that has been attached to the timeline.
OmPlrGetClipFullName	Returns full name information about attached clips.
OmPlrGetClipPath	Find out the pathname for a clip that has already been attached to the timeline.
OmPlrGetTcgInsertion	Get the insertion settings of the internal timecode generator.
OmPlrGetTcgMode	Get the mode setting of the internal timecode generator.
OmPlrLoop	Set the first and last timeline positions to establish a playback looping mode.
OmPlrSetClipData	Used to change properties of a clip as attached to the timeline.
OmPlrSetMaxPos	Set the Maximum allowed position on the timeline.
OmPlrSetMaxPosMax	Set the Maximum allowed position on the timeline at 1 beyond the end of last clip.
OmPlrSetMinMaxPosToClip	Set the Minimum and Maximum allowed positions on the timeline using the start and end of “this” clip.
OmPlrSetMinMaxPosToClip	Set the Minimum allowed position on the timeline.
OmPlrSetMinPosMin	Set the Minimum allowed position on the timeline to the first frame of the first clip.
OmPlrSetTcgData	Set the data for the internal timecode generator.
OmPlrSetTcgInsertion	Enable insertion of the output of the internal timecode generator into video play or record streams.

Function	Description
OmPlrSetTcgMode	Set the mode for the internal timecode generator.
OmPlrSetClipVfc	Used to modify the video frame conversion mode of clip on the timeline.

Timeline Function Discussion

This section describes the Spectrum timeline. It concentrates on the functions that attach clips to the timeline and that manipulate clips after they have been attached to the timeline.

A Player can deal with a clip or with a set of clips. When a Player begins to "play a clip," it does that indirectly. The mechanism that supports the playing of the clips is the **Timeline**. Each Player has one and only one timeline object. The timeline is very important to the Player object; a measure of the importance is that often conversations about the Player's timeline will blur the distinction between the Player and its timeline. When you see the Spectrum system "playing a clip," the Player object is busy "playing" its timeline; the clip just happens to be attached to the timeline and gets to go along for the ride.

A timeline is a list of clips that are played as a set, one after the other. An analogy for the timeline is a freight train of flat cars. The Player's attach function "loads a clip onto a flat car" and then the Player's play function causes the whole train to start moving, bringing each of the clips past the person standing next to the tracks. The person sees one clip after another. Each clip is attached to the next and to the previous. All the clips move together as a unit. And the train can go at various speeds, can stop at any point, and can go backwards at various speeds.

This train analogy corresponds to a typical simple use of the Player to play a set of clips. To play a set of clips, you would:

- Remove any previous clips from the timeline.
- Issue a stop function to prevent any premature motion.
- Attach the first clip to the timeline object.

- Attach the next clip after the first.
- Attach the rest of the clips, appending each to the end of the set.
- Set the timeline min parameter to be at the start of the first clip.
- Set the timeline max parameter to be at the end of the last clip.
- Set the timeline position to be at the start of the first clip.
- Tell the timeline to starting playing at a 1X forward speed.
- Sit back and watch the clips play out.

Do not carry this analogy too far (we do not have any conductors or cabooses inside the Spectrum MediaDirector, but you can sometimes hear the puffing of a steam engine if you put your ear up close to the MediaDirector).

The Spectrum timeline concept has many of the same properties as the process of editing film:

- Each clip on the timeline is attached to the clip before it and to the clip after it — with no gaps and no overlaps.
- Clips can be added anywhere in the set of clips.
- A clip is added to the set of clips by specifying the existing clip to follow the new clip.
- If a clip is added in the middle of the timeline, then a gap is automatically made to receive the new clip by moving the other clips.
- Clips can be individually removed from the timeline.
- When a clip is removed, the resulting gap is automatically closed up.
- When the timeline is played back, the clips are seen in the same order as their attachment to the timeline.
- Only one play function is needed to set the whole set of clips into motion.
- The set of clips may be watched in either direction at any speed.
- The set of clips remains intact as a set on the timeline even after you have watched the set once. The set can be viewed innumerable times.

The Spectrum timeline concept extends beyond film editing to include the following:

- You can add or delete clips from the timeline even while the timeline is in motion. Again any resulting gaps or overlaps are automatically removed.
- The start and end of each clip can be adjusted even after the clip has been attached to the timeline. Any resulting gaps or overlaps are automatically removed.
- The clip can be attached with start and end points that are beyond the end of the recorded media. In that case, black (silence) is shown for the unrecorded portions.
- The timeline motion can extend beyond the end of the last attached clip. Black (silence) will be shown when the timeline is beyond the limit. The timeline can also be started before the start of any attached clip; again black will be shown for the portion of the timeline that has no attached media.
- The timeline can be set up for looping – which means that one section of the timeline will be shown repeatedly. After the last frame of the looped section is played, the next viewed frame will occur at the start of the section. The looping motion can go in either direction and can go at any speed.
- The timeline is also used to create new clips and to record media into the new clips.

Timecode Discussion

The section describes how the Spectrum system processes timecode. When the Spectrum system records a clip, it can record timecode data along with the video, audio and other information. Typically the recording of a clip is done with an Spectrum MediaPort. The MediaPort has connections for Longitudal Timecode (LTC); it can also process Vertical Interval Timecode (VITC) that is embedded in the incoming video signal. In either case, the timecode signal includes 32 bits of *user bits* in addition to the 32 bits of time information. The Spectrum system records all bits of the incoming timecode signal.

An overview of Spectrum timecode capability is listed below. The Spectrum system:

- Is able to record incoming timecode data (time data plus user bits) and save the information within the clip's media files.
- Is able to reproduce the timecode data when playing back the clip.
- Provides an internal timecode generator that can provide a substitute source of timecode while recording a clip.
- Provides an internal timecode generator that can provide a substitute source of timecode while playing back a clip.
- Provides API functions for controlling the internal timecode generator.
- Provides an API function that can be used to read the timecode data while the clip is being played back, recorded, cued for playback or cued for recording.
- Provides an API function that can be used to position a clip to a given timecode position.

How does timecode relate to the position returned by OmPlrGetPlayerStatus

There is no direct relationship. The timeline position is a count of frames that doesn't have any relationship to the clip's timecode. In addition, the IN/OUT data for a clip that is returned by the **OmPlrClipGetInfo** command is not related to the recorded timecode values. Any relationship was indirectly established by the application using API commands to control the MediaDirector and to process time data.

Handling of the timecode signal during typical recording and playback

Typically the timecode signal that is fed into the MediaPort will be automatically recorded when a clip is being recorded. The Spectrum configuration determines whether the incoming signal is VITC (taken from the SDI input) or LTC (taken from the LTC input). On the other hand, the un-typical case records a timecode signal that is generated from an internal timecode generator; that case is explained below.

The recorded timecode is stored somewhere in the media files; the key point is that the recorded timecode is not readily available as "computer readable" data for the MediaDirector's "player" object.

During clip playback, typically the recorded timecode signal will be retrieved and will be made available at the MediaPort outputs. The signal is presented on the LTC output port and, if the device is configured for VITC, it will also be presented simultaneously on the SDI port. In the non-typical case, the output of an internal generator can be substituted for the media signal; this output provides the timecode signal values output by the MediaPort.

When the playback is at speeds other than 1X, the LTC signal is still generated at a 1X rate (ie, 80 bits of data every 33 or 40 milliseconds). Please note:

- At speeds faster than 1X, timecode values will be skipped.
- At speeds less than 1X, timecode values will be repeated.

- At still speed, a constant timecode value is endlessly repeated.
- During reverse motion, the LTC timecode is presented with low bits first (as it was in forward motion).

NOTE: If no external LTC timecode is provided during a recording operation, the timecode recorded with the media will be a special pattern indicating "invalid timecode." If configured for VITC and no external timecode is provided during a record operation, the timecode recorded with the media will be a special pattern indicating "invalid timecode."

Use of the internal Timecode Generator while recording

Inside the Spectrum MediaDirector is a software based timecode generator; each Player object has its own independent generator. During the recording of a clip, the output of this generator can be used in place of the incoming timecode to provide an alternative source of the timecode signal for recording with the media. Since the generator is a software entity, it is available for use both in VITC and LTC configurations. It generates timecode at a 1X rate. The *OmPlrSetTcgInsertion* command is used to enable inserting of the timecode generator data into the record stream.

The internal timecode generator can be controlled to stop or to increment constantly at a 1X rate by use of the *OmPlrSetTcgMode* function. The starting timecode (and user bits) for the generator are set using the *OmPlrSetTcgData* function.

The **OmPlrSetTcgMode** function can be used to put the generator into the following modes for recording:

- **omPlrTcgModeHold**
- **omPlrTcgModeFreeRun**
- **omPlrTcgModeLockedTimeline**
- **omPlrTcgModeLockedClip**
- **omPlrTcgModeLockedRefVitc**

Refer to *OmPlrSetTcgMode* for additional information on each generator modes.

Capabilities of the software timecode generator during playback

Just as the internal generator can be used to substitute for the incoming timecode signal during record operations, it can also be used during playback operations to provide a timecode signal that is substituted for the one that was recorded with the media. The following generator modes apply during playback:

- **omPlrTcgModeHold**
- **omPlrTcgModeFreeRun**
- **omPlrTcgModeLockedTimeline**
- **omPlrTcgModeLockedClip**
- **omPlrTcgModeLockedClipTc**
- **omPlrTcgModeLockedRefVitc**

NOTE: If the generator mode is omPlrTcgModeFreeRun, then the generator will provide timecode at a 1X rate independent of the speed of the timeline playback. In the two Locked modes, the generator will provide timecode at a 1x rate that will repeat or skip as necessary in order to match the timeline position.

Refer to *OmPlrSetTcgMode* for additional information on each generator modes.

Capabilities of the software timecode reader during record and playback

Inside the MediaDirector there is an internal software based timecode reader for each player object. The data from the timecode reader is retrieved by the **OmPlrGetTime** function. The MediaDirector provides a snapshot of its internal frame counter when it provides the reader value; this allows the application to tie a particular timecode reading back to house time. The internal reader also reads the user bits that are part of the timecode.

During record operations, the reader can sniff the timecode stream that is being recorded. The stream will either be the incoming timecode signal or the substitute signal that is generated by the internal timecode generator. The reader will start reading the input timecode when the Player is put into the cue record mode.

The internal timecode reader can be used during playback operation to sniff the timecode stream that is being sent to the MediaPort. The timecode can be read if the clip is loaded and if the Player is not in stop mode. Timecode can be read in pause/cued mode and also at all speeds of forward or reverse play. The timecode reported by the reader is adjusted for pipeline delays so that it represents the timecode at the MediaPort input or output, assuming a media speed of 1X.

API commands that access the software timecode reader

- *OmPlrGetTime*
- *OmPlrGoToTimecode*

API commands that set up the software timecode generator

- *OmPlrSetTcgData*
- *OmPlrSetTcgMode*
- *OmPlrSetTcgInsertion*
- *OmPlrGetTcgMode*
- *OmPlrGetTcgInsertion*

OmPlrAttach

Description

Used to attach a clip to the timeline. Also used to create a new clip.

```
OmPlrError OmPlrAttach(
OmPlrHandle PlayerHandle,
const TCHAR *pClipName,
uint clipIn,
uint clipOut,
OmPlrClipHandle hPutBeforeThisClip,
OmPlrShiftMode shift,
OmPlrClipHandle *pClipAttachHandle);
```

Parameters

Parameter	Description
PlayerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipname	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy". The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
clipIn	A number that identifies the first frame of this clip to be shown (inclusive). The special value **omPlrClipDefaultIn** can be used; this value is defined in omplrdefs.h. It means that the "Default In" frame value found in the stored version of the clip will be used for the *clipIn* value.
clipOut	A number that identifies the frame after the last frame of this clip to be shown. The special value **omPlrClipDefaultOut** can be used; this value is defined in omplrdefs.h. It means that the "Default Out" frame value found in the stored version of the clip will be used for the **clipOut** value.
hPutBeforeThisClip	A numeric value that is 0 or is the **OmPlrClipHandle** of an already attached clip. Typically a clip will be attached at the end of the timeline after all clips that were previously attached. But you can also attach a clip in front of other clips that are already attached to the timeline. Use a value of 00 to attach at the end (i.e., to append). Use the **OmPlrClipHandle** value of the already attached clip if you want to attach this clip in front of the already attached clip.

Parameter	Description
shift	A numeric value of **omPlrShiftModeBefore**, **omPlrShiftModeAfter**, or **omPlrShiftModeAuto**; these are defined in the file omplrdefs.h. The value of **omPlrShiftModeOthers** doesn't apply when attaching clips and should not be used. When a clip is attached in the middle of a set of already attached clips, a hole has to be opened up in the existing clips to accept the newly attached clip; the hole will be large enough so that the newly attached clip will align with the previously attached clips. The *shift* parameter is used to indicate your desire on how the hole should be created. The value of **omPlrShiftModeAfter** causes the hole to be created by moving clips after the attachment point so that these clips will be shown later. The value of **omPlrShiftModeBefore** causes the hole to be created by shifting the previously attached clips before the attachment point to earlier locations on the timeline. A value of **omPlrShiftModeAuto** will use one of the two modes of **omPlrShiftModeAfter** or **omPlrShiftModeBefore**; the choice will be made to avoid changing the media at the current timeline position.
pClipAttachHandle	A pointer used to return a numeric value that has been assigned by the Player to the clip that has been attached to the timeline. This number is returned to the caller to be used as a handle; the handle will be needed as a parameter for future calls that alter this clip on the timeline or that remove it from the timeline. The clip attach handle value will not change while the clip is attached to the timeline. The set of clip attach handle values at any one moment of time will not have any duplications; a particular handle value will often be quickly reused after a **OmPlrDetach** command. Note: a valid clip attach handle will never have a value of 00.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

The function will fail and an error of **omPlrClipTooLong** will be returned if **clipOut** is less than clipIn.

Remarks

The values of **clipIn** and **clipOut** are tied to the frame numbering of the clip; the numbers are not timeline numbers. The definition of **clipIn** and **clipOut** in terms of first frame seen and last frame seen assumes the typical case of forward play. A more complete definition would be to say that no frame with a number less than **clipIn** will be seen and that no frame with a number the same as **clipOut** or larger will be seen.

The identifying number assigned to other clips attached to the timeline may change after this function has executed; this number is used as an argument for functions such as **OmPlrGetClipAtNum**.

The **clipIn** and **clipOut** values can be adjusted after the clip has been attached using the function **OmPlrSetClipData**. **clipIn** and **clipOut** can define a clip that starts or ends outside of the recorded video; in that case you get black output for the un-recorded parts of the clip.

When creating a clip, the difference of "**clipOut** – **clipIn**" will determine the maximum amount of material that will be recorded into this clip. No material will be recorded into this clip if **clipOut** = **clipIn**. The arguments **clipOut** and **clipOut** can be any value as long as **clipIn** is not more than **clipOut**. If the difference of **clipOut** – **clipIn** defines too large of a clip, the function will fail with the error **omPlrClipTooLong**; the limit is slightly more than 24 hours.

omPlrShiftModeBefore can cause the timeline to start at a negative value. This is valid (timeline position is a signed number) but typically the timeline starts at a value of 00.

The values **omPlrShiftModeBefore**, **omPlrShiftModeAfter**, and **omPlrShiftModeAuto** are part of an enumeration that is defined in the file omplrdefs.h.

When loading clips where the **clipName** argument is not a fully qualified clip name, the first clip that matches any of the extensions in the clip list will be the one clip that is loaded. When using **OmPlrAttach** to create a new clip when the **clipName** parameter is not a fully qualified clip name, the clip is given the extension that is first in the current clip extension list. For instance if the extension list is ".mxf.mov" then a "fred.mxf" clip will be created if the command is **OmPlrCreate** for a clipname of "fred". Note that only some video formats can be recorded as MXF clips; the **OmPlrCreate** function will fail if asked to create a .mxf clip in a format that is not supported.

The clip name *SpectrumEmptyBlackClip* is reserved for a special case of attaching a section of black (and silent) media to the timeline. The properties of this special clip name are:

- The clip does not need to exist in the file system. The clip name will not appear in a list of clips. No files are created in the file system for this clip.
- The clip can be attached for any length, including 0.
- The clip can be attached even if the player is configured for PLAY ONLY.
- Attaching the clip does not trigger the **Add** callback. And the **Delete** callback is not triggered when this clip is detached.

CAUTION: Be aware that attaching a clip to the timeline does NOT alter the maximum and minimum points for the timeline. Typically, you will also want to alter these values, so that the timeline is allowed to march across the clip and show it to you. Setting the timeline maximum point is also important when recording.

Also be aware that an empty timeline can still have a motion state, such as 10X forward play. Attaching a clip to the timeline does not affect the motion state of the timeline, so that if a clip is attached to an empty timeline that happens to have a 10X forward motion state, the clip will instantly take off playing at that speed.

If you should intend to open an existing clip but provide the wrong clip name, you will have just created a clip of the wrong name. No error message is given because the MediaDirector assumes that your intent is to create a new clip. The wrongly named clip will quietly disappear from the file system if you detach it without recording anything into it.

When you are attaching a clip for playback that is currently being recorded, you must wait until at least 60 frames have been recorded before doing the attach for playback; otherwise black will be shown in place of the clip

See also

Player Function Discussion, *Clip Function Discussion*, *Timeline Function Discussion*, *OmPlrSetMaxPos*, *OmPlrSetMinPos*, *OmPlrOpen*

OmPlrAttach1

Description

Another version of *OmPlrAttach*. This one adds a modified track match string.

```
OmPlrError OmPlrAttach1(
OmPlrHandle plrHandle,
const TCHAR *pClipName,
uint clipIn,
uint clipOut,
OmPlrClipHandle beforeClipHandle,
OmPlrShiftMode shiftMode,
const char *pTrackMatch,
OmPlrClipHandle *pClipHandle);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
pClipname	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that “Harry” is not the same clip as “haRRy”. The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
clipIn	A number that identifies the first frame of this clip to be shown (inclusive). The special value **omPlrClipDefaultIn** can be used; this value is defined in omplrdefs.h. It means that the “Default In” frame value found in the stored version of the clip will be used for the **clipIn** value.
clipOut	A number that identifies the frame after the last frame of this clip to be shown. The special value **omPlrClipDefaultOut** can be used; this value is defined in omplrdefs.h. It means that the “Default Out” frame value found in the stored version of the clip will be used for the **clipOut** value.
beforeClipHandle	A numeric value that is 0 or is the **OmPlrClipHandle** of an already attached clip. Typically a clip will be attached at the end of the timeline after all clips that were previously attached. But you can also attach a clip in front of other clips that are already attached to the timeline. Use a value of 00 to attach at the end (i.e., to append). Use the **OmPlrClipHandle** value of the already attached clip if you want to attach this clip in front of the already attached clip.

Parameter	Description
shiftMode	A numeric value of **omPlrShiftModeBefore**, **omPlrShiftModeAfter**, or **omPlrShiftModeAuto**; these are defined in the file omplrdefs.h. The value of **omPlrShiftModeOthers** doesn't apply when attaching clips and should not be used. When a clip is attached in the middle of a set of already attached clips, a hole has to be opened up in the existing clips to accept the newly attached clip; the hole will be large enough so that the newly attached clip will align with the previously attached clips. The **shift** parameter is used to indicate your desire on how the hole should be created. The value of **omPlrShiftModeAfter** causes the hole to be created by moving clips after the attachment point so that these clips will be shown later. The value of **omPlrShiftModeBefore** causes the hole to be created by shifting the previously attached clips before the attachment point to earlier locations on the timeline. A value of **omPlrShiftModeAuto** will use one of the two modes of **omPlrShiftModeAfter** or **omPlrShiftModeBefore**; the choice will be made to avoid changing the media at the current timeline position.
pTrackMatch	A pointer to UTF-8 characters that form a "C" style string. The string is used for selecting and reordering the tracks in a clip for loading.
pClipHandle	A pointer used to return a numeric value that has been assigned by the Player to the clip that has been attached to the timeline. This number is returned to the caller to be used as a handle; the handle will be needed as a parameter for future calls that alter this clip on the timeline or that remove it from the timeline. The clip attach handle value will not change while the clip is attached to the timeline. The set of clip attach handle values at any one moment of time will not have any duplications; a particular handle value will often be quickly reused after a OmPlrDetach command. Note: a valid clip attach handle will never have a value of 00.

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

The function will fail and an error of **omPlrClipTooLong** will be returned if **clipOut** is less than clipIn.

Remarks

The track match string must start with "audioM:" followed by a track match string. An example string is "audioM:token1/token2, token3, token4".A track match string is used when attaching a clip. It is compared against user data in the clip to select which tracks from the clip to map to player channels.

The string has the following format: token1/token2/token3,token4/token5 Matching occurs between a token and clip track user data value for key "ovn_trackmatch". See *OmPlrClipSetStartTimecode* and *OmPlrClipSetTrackUserData*. For the above track match string, the system first tries to find a clip track labeled token1, token2 or token3 in that order. The first match will be used. Then it will look for a clip track labeled token4 or token5. Again, the first match will be used. There are some special token names. The special name silenceX where X is a number will cause X channels of silence to be used. For example: a match string such as "english/silence2" will first look for a clip track labeled "english" and failing that will use 2 channels of silence. The special name **tracknumX** where X is a number will always match clip track number X.

The track match string is composed of UTF-8 characters. The typical tokens just use ASCII characters in the range of 0x20 to x07F; these typical characters happen to be valid UTF-8. The tokens in this string must match the tokens that are written into the track user data of the clip. The Spectrum SystemManager uses UTF-8 encoding in the TrackKeyValueDefs.txt file that it creates; this file is the source of the track tag tokens. An application that uses UTF-16 for most internal strings must be careful to use UTF-8 characters for this string.

The track match string pointer may be 0 or the track match string may be zero length.

See also

OmPlrAttach, *OmPlrOpen*, *OmPlrClipSetStartTimecode*, *OmPlrClipSetTrackUserData*, *Clip Name as Function Argument*

OmPlrAttach2

Description

Another version of *OmPlrAttach*. This one adds a video frame conversion argument.

```
OmPlrError OmPlrAttach2(
OmPlrHandle plrHandle,
const TCHAR *pClipName,
uint clipIn,
uint clipOut,
OmPlrClipHandle beforeClipHandle,
OmPlrShiftMode shiftMode,
const char *pTrackMatch,
OmPlrVideoFrameConvert vfc,
OmPlrClipHandle *pClipHandle);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
pClipname	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that “Harry” is not the same clip as “haRRy”. The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
clipIn	A number that identifies the first frame of this clip to be shown (inclusive). The special value **omPlrClipDefaultIn** can be used; this value is defined in omplrdefs.h. It means that the “Default In” frame value found in the stored version of the clip will be used for the **clipIn** value.
clipOut	A number that identifies the frame after the last frame of this clip to be shown. The special value **omPlrClipDefaultOut** can be used; this value is defined in omplrdefs.h. It means that the “Default Out” frame value found in the stored version of the clip will be used for the **clipOut** value.

Parameter	Description
beforeClipHandle	A numeric value that is 0 or is the **OmPlrClipHandle** of an already attached clip. Typically a clip will be attached at the end of the timeline after all clips that were previously attached. But you can also attach a clip in front of other clips that are already attached to the timeline. Use a value of 00 to attach at the end (i.e., to append). Use the **OmPlrClipHandle** value of the already attached clip if you want to attach this clip in front of the already attached clip.
shiftMode	A numeric value of **omPlrShiftModeBefore**, **omPlrShiftModeAfter**, or **omPlrShiftModeAuto**; these are defined in the file omplrdefs.h. The value of **omPlrShiftModeOthers** doesn't apply when attaching clips and should not be used. When a clip is attached in the middle of a set of already attached clips, a hole has to be opened up in the existing clips to accept the newly attached clip; the hole will be large enough so that the newly attached clip will align with the previously attached clips. The **shift** parameter is used to indicate your desire on how the hole should be created. The value of **omPlrShiftModeAfter** causes the hole to be created by moving clips after the attachment point so that these clips will be shown later. The value of **omPlrShiftModeBefore** causes the hole to be created by shifting the previously attached clips before the attachment point to earlier locations on the timeline. A value of **omPlrShiftModeAuto** will use one of the two modes of **omPlrShiftModeAfter** or **omPlrShiftModeBefore**; the choice will be made to avoid changing the media at the current timeline position.
pTrackMatch	A pointer to UTF-8 characters that form a "C" style string. The string is used for selecting and reordering the tracks in a clip for loading.
vfc	Gives the video frame conversion (aspect ratio conversion or ARC) to be used. See *Definition of Fields in the OmPlrVideoFrameConvert Enum* for details.
pClipHandle	A pointer used to return a numeric value that has been assigned by the Player to the clip that has been attached to the timeline. This number is returned to the caller to be used as a handle; the handle will be needed as a parameter for future calls that alter this clip on the timeline or that remove it from the timeline. The clip attach handle value will not change while the clip is attached to the timeline. The set of clip attach handle values at any one moment of time will not have any duplications; a particular handle value will often be quickly reused after a OmPlrDetach command. Note: a valid clip attach handle will never have a value of 00.

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

The function will fail and an error of **omPlrClipTooLong** will be returned if **clipOut** is less than clipIn.

Definition of Fields in the OmPlrVideoFrameConvert Enum

The enum **OmPlrVideoFrameConvert** is defined in **omplrdefs.h** as:

```
enum OmPlrVideoFrameConvert(
   omPlrVfcUnknown, //unspecified conversion
   omPlrVfcNone, //no conversion
   omPlrVfcExternal0,  //external converter, 0 frame delay,
```

```
omPlrVfcExternal1, //external converter, 1 frame delay,
omPlrVfcExternal2, //external converter, 2 frame delay,
omPlrVfcExternal3, //external converter, 3 frame delay,
omPlrVfcExternal4, //external converter, 4 frame delay,
omPlrVfcPillar, //4x3 => 16x9 (black on sides),
omPlrVfcCrop, //4x3 => 16x9 (chop top and bottom, some black
sides),
      //16x9 => 4x3 (chop sides, some black top/bottom)
OmPlrVfcLetter, //16x9 => 4x3(black top/bottom),
OmPlrVfcFull, //16x9 => 4x3 (chop sides, no black top.bottom),
      //4x3 => 16x9 (chop top/bottom, no black sides)
OmPlrVfcAnamorphic, //4x3 => 16x9 (stretch)
      //16x9 => 4x3 (squeeze)
```

Remarks

The track match string must start with "audioM:" followed by a track match string. An example string is "audioM:token1/token2, token3, token4".A track match string is used when attaching a clip. It is compared against user data in the clip to select which tracks from the clip to map to player channels.

The string has the following format: token1/token2/token3,token4/token5 Matching occurs between a token and clip track user data value for key "ovn_trackmatch". See *OmPlrClipSetStartTimecode* and *OmPlrClipSetTrackUserData*. For the above track match string, the system first tries to find a clip track labeled token1, token2 or token3 in that order. The first match will be used. Then it will look for a clip track labeled token4 or token5. Again, the first match will be used. There are some special token names. The special name silenceX where X is a number will cause X channels of silence to be used. For example: a match string such as "english/silence2" will first look for a clip track labeled "english" and failing that will use 2 channels of silence. The special name **tracknumX** where X is a number will always match clip track number X.

The track match string is composed of UTF-8 characters. The typical tokens just use ASCII characters in the range of 0x20 to x07F; these typical characters happen to be valid UTF-8. The tokens in this string must match the tokens that are written into the track user data of the clip. The Spectrum SystemManager uses UTF-8 encoding in the TrackKeyValueDefs.txt file that it creates; this file is the source of the track tag tokens. An application that uses UTF-16 for most internal strings must be careful to use UTF-8 characters for this string.

The track match string pointer may be 0 or the track match string may be zero length.

See also

OmPlrAttach, *OmPlrOpen*, *OmPlrClipSetStartTimecode*, *OmPlrClipSetTrackUserData*, *Clip Name as Function Argument*

OmPlrAttach3

Description

Another version of *OmPlrAttach*. This one starts with *OmPlrAttach2* and adds an AFD and secondary audio mix select.

```
OmPlrError OmPlrAttach3(
OmPlrHandle plrHandle,
const TCHAR *pClipName,
uint clipIn,
uint clipOut,
```

```
                OmPlrClipHandle beforeClipHandle,
                OmPlrShiftMode shiftMode,
                const char *pTrackMatch,
                OmPlrVideoFrameConvert vfc,
                uint afd,
                uint secAudMixSel,
                OmPlrClipHandle *pClipHandle);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
pClipname	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy". The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
clipIn	A number that identifies the first frame of this clip to be shown (inclusive). The special value **omPlrClipDefaultIn** can be used; this value is defined in omplrdefs.h. It means that the "Default In" frame value found in the stored version of the clip will be used for the **clipIn** value.
clipOut	A number that identifies the frame after the last frame of this clip to be shown. The special value **omPlrClipDefaultOut** can be used; this value is defined in omplrdefs.h. It means that the "Default Out" frame value found in the stored version of the clip will be used for the **clipOut** value.
beforeClipHandle	A numeric value that is 0 or is the **OmPlrClipHandle** of an already attached clip. Typically a clip will be attached at the end of the timeline after all clips that were previously attached. But you can also attach a clip in front of other clips that are already attached to the timeline. Use a value of 00 to attach at the end (i.e., to append). Use the **OmPlrClipHandle** value of the already attached clip if you want to attach this clip in front of the already attached clip.

Parameter	Description
shiftMode	A numeric value of **omPlrShiftModeBefore**, **omPlrShiftModeAfter**, or **omPlrShiftModeAuto**; these are defined in the file omplrdefs.h. The value of **omPlrShiftModeOthers** doesn't apply when attaching clips and should not be used. When a clip is attached in the middle of a set of already attached clips, a hole has to be opened up in the existing clips to accept the newly attached clip; the hole will be large enough so that the newly attached clip will align with the previously attached clips. The **shift** parameter is used to indicate your desire on how the hole should be created. The value of **omPlrShiftModeAfter** causes the hole to be created by moving clips after the attachment point so that these clips will be shown later. The value of **omPlrShiftModeBefore** causes the hole to be created by shifting the previously attached clips before the attachment point to earlier locations on the timeline. A value of **omPlrShiftModeAuto** will use one of the two modes of **omPlrShiftModeAfter** or **omPlrShiftModeBefore**; the choice will be made to avoid changing the media at the current timeline position.
pTrackMatch	A pointer to UTF-8 characters that form a "C" style string. The string is used for selecting and reordering the tracks in a clip for loading.
vfc	Gives the video frame conversion (aspect ratio conversion or ARC) to be used. See *Definition of Fields in the OmPlrVideoFrameConvert Enum* for details.
afd	Active format description, coded as in section 9 of SMPTE 2016-1. The AFD overrides any other AFD associated with the clip.
secAudMixSel	Specifies an index used to look up an audio mix mode to convert the main audio to the secondary audio output.
pClipHandle	A pointer used to return a numeric value that has been assigned by the Player to the clip that has been attached to the timeline. This number is returned to the caller to be used as a handle; the handle will be needed as a parameter for future calls that alter this clip on the timeline or that remove it from the timeline. The clip attach handle value will not change while the clip is attached to the timeline. The set of clip attach handle values at any one moment of time will not have any duplications; a particular handle value will often be quickly reused after a OmPlrDetach command. Note: a valid clip attach handle will never have a value of 00.

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

The function will fail and an error of **omPlrClipTooLong** will be returned if **clipOut** is less than clipIn.

Definition of Fields in the OmPlrVideoFrameConvert Enum

The enum **OmPlrVideoFrameConvert** is defined in **omplrdefs.h** as:

```
enum OmPlrVideoFrameConvert(
   omPlrVfcUnknown, //unspecified conversion
   omPlrVfcNone, //no conversion
   omPlrVfcExternal0,  //external converter, 0 frame delay,
```

```
omPlrVfcExternal1, //external converter, 1 frame delay,
omPlrVfcExternal2, //external converter, 2 frame delay,
omPlrVfcExternal3, //external converter, 3 frame delay,
omPlrVfcExternal4, //external converter, 4 frame delay,
omPlrVfcPillar, //4x3 => 16x9 (black on sides),
omPlrVfcCrop, //4x3 => 16x9 (chop top and bottom, some black
sides),
      //16x9 => 4x3 (chop sides, some black top/bottom)
OmPlrVfcLetter, //16x9 => 4x3(black top/bottom),
OmPlrVfcFull, //16x9 => 4x3 (chop sides, no black top.bottom),
      //4x3 => 16x9 (chop top/bottom, no black sides)
OmPlrVfcAnamorphic, //4x3 => 16x9 (stretch)
      //16x9 => 4x3 (squeeze)
```

Remarks

The track match string must start with "audioM:" followed by a track match string. An example string is "audioM:token1/token2, token3, token4".A track match string is used when attaching a clip. It is compared against user data in the clip to select which tracks from the clip to map to player channels.

The string has the following format: token1/token2/token3,token4/token5 Matching occurs between a token and clip track user data value for key "ovn_trackmatch". See *OmPlrClipSetStartTimecode* and *OmPlrClipSetTrackUserData*. For the above track match string, the system first tries to find a clip track labeled token1, token2 or token3 in that order. The first match will be used. Then it will look for a clip track labeled token4 or token5. Again, the first match will be used. There are some special token names. The special name silenceX where X is a number will cause X channels of silence to be used. For example: a match string such as "english/silence2" will first look for a clip track labeled "english" and failing that will use 2 channels of silence. The special name **tracknumX** where X is a number will always match clip track number X.

The track match string is composed of UTF-8 characters. The typical tokens just use ASCII characters in the range of 0x20 to x07F; these typical characters happen to be valid UTF-8. The tokens in this string must match the tokens that are written into the track user data of the clip. The Spectrum SystemManager uses UTF-8 encoding in the TrackKeyValueDefs.txt file that it creates; this file is the source of the track tag tokens. An application that uses UTF-16 for most internal strings must be careful to use UTF-8 characters for this string.

The track match string pointer may be 0 or the track match string may be zero length.

See also

OmPlrAttach, *OmPlrOpen*, *OmPlrClipSetStartTimecode*, *OmPlrClipSetTrackUserData*, *Clip Name as Function Argument*

OmPlrAttach4

Description

Another version of *OmPlrAttach*. This one starts with *OmPlrAttach3* and adds the list of caption or subtitle files to insert during playback. This list overrides any other caption or subtitle files associated with the clip. The list of files is given as a single string formatted as a JSON array, for example: "[{\"file\":\"cap_eng.scc\"},{\"file\":\"cap_spa.scc\"}]". Filenames can be absolute or relative. If relative then the configured subtitle directory is used as a base location. Use an empty string to not override the associations set in the player configuration.

```
OmPlrError OmPlrAttach4A(
OmPlrHandle plrHandle,
```

```
const CHAR *pClipName,
uint clipIn,
uint clipOut,
OmPlrClipHandle beforeClipHandle,
OmPlrShiftMode shiftMode,
const char *pTrackMatch,
OmPlrVideoFrameConvert vfc,
uint afd,
uint secAudMixSel,
const char *pSubtitleFileList,
OmPlrClipHandle *pClipHandle);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
pClipname	A pointer to a null-terminated string that gives the clip name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. The clip name is case sensitive, so that "Harry" is not the same clip as "haRRy". The argument can specify either a clip or the combination of an absolute path and a clip; see *Clip Name as Function Argument* for more details. The size of the argument is limited to 512.
clipIn	A number that identifies the first frame of this clip to be shown (inclusive). The special value **omPlrClipDefaultIn** can be used; this value is defined in omplrdefs.h. It means that the "Default In" frame value found in the stored version of the clip will be used for the **clipIn** value.
clipOut	A number that identifies the frame after the last frame of this clip to be shown (exclusive). The special value **omPlrClipDefaultOut** can be used; this value is defined in omplrdefs.h. It means that the "Default Out" frame value found in the stored version of the clip will be used for the **clipOut** value.
beforeClipHandle	A numeric value that is 0 or is the **OmPlrClipHandle** of an already attached clip. Typically a clip will be attached at the end of the timeline after all clips that were previously attached. But you can also attach a clip in front of other clips that are already attached to the timeline. Use a value of 00 to attach at the end (i.e., to append). Use the **OmPlrClipHandle** value of the already attached clip if you want to attach this clip in front of the already attached clip.

Parameter	Description
shiftMode	A numeric value of **omPlrShiftModeBefore**, **omPlrShiftModeAfter**, or **omPlrShiftModeAuto**; these are defined in the file omplrdefs.h. The value of **omPlrShiftModeOthers** doesn't apply when attaching clips and should not be used. When a clip is attached in the middle of a set of already attached clips, a hole has to be opened up in the existing clips to accept the newly attached clip; the hole will be large enough so that the newly attached clip will align with the previously attached clips. The **shift** parameter is used to indicate your desire on how the hole should be created. The value of **omPlrShiftModeAfter** causes the hole to be created by moving clips after the attachment point so that these clips will be shown later. The value of **omPlrShiftModeBefore** causes the hole to be created by shifting the previously attached clips before the attachment point to earlier locations on the timeline. A value of **omPlrShiftModeAuto** will use one of the two modes of **omPlrShiftModeAfter** or **omPlrShiftModeBefore**; the choice will be made to avoid changing the media at the current timeline position.
pTrackMatch	A pointer to UTF-8 characters that form a "C" style string. The string is used for selecting and reordering the tracks in a clip for loading.
vfc	Gives the video frame conversion (aspect ratio conversion or ARC) to be used. See *Definition of Fields in the OmPlrVideoFrameConvert Enum* for details.
afd	Active format description, coded as in section 9 of SMPTE 2016-1. The AFD overrides any other AFD associated with the clip.
secAudMixSel	Specifies an index used to look up an audio mix mode to convert the main audio to the secondary audio output.
pSubtitleFileList	The optional caption/subtitle file list. See *About pSubtitleFileList*.
pClipHandle	A pointer used to return a numeric value that has been assigned by the Player to the clip that has been attached to the timeline. This number is returned to the caller to be used as a handle; the handle will be needed as a parameter for future calls that alter this clip on the timeline or that remove it from the timeline. The clip attach handle value will not change while the clip is attached to the timeline. The set of clip attach handle values at any one moment of time will not have any duplications; a particular handle value will often be quickly reused after a OmPlrDetach command. Note: a valid clip attach handle will never have a value of 00.

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

The function will fail and an error of **omPlrClipTooLong** will be returned if **clipOut** is less than clipIn.

Definition of Fields in the OmPlrVideoFrameConvert Enum

The enum **OmPlrVideoFrameConvert** is defined in **omplrdefs.h** as:

```
enum OmPlrVideoFrameConvert(
   omPlrVfcUnknown, //unspecified conversion
   omPlrVfcNone, //no conversion
   omPlrVfcExternal0,  //external converter, 0 frame delay,
```

```
omPlrVfcExternal1, //external converter, 1 frame delay,
omPlrVfcExternal2, //external converter, 2 frame delay,
omPlrVfcExternal3, //external converter, 3 frame delay,
omPlrVfcExternal4, //external converter, 4 frame delay,
omPlrVfcPillar, //4x3 => 16x9 (black on sides),
omPlrVfcCrop, //4x3 => 16x9 (chop top and bottom, some black
sides),
      //16x9 => 4x3 (chop sides, some black top/bottom)
OmPlrVfcLetter, //16x9 => 4x3(black top/bottom),
OmPlrVfcFull, //16x9 => 4x3 (chop sides, no black top.bottom),
      //4x3 => 16x9 (chop top/bottom, no black sides)
OmPlrVfcAnamorphic, //4x3 => 16x9 (stretch)
      //16x9 => 4x3 (squeeze)
```

Remarks

The track match string must start with "audioM:" followed by a track match string. An example string is "audioM:token1/token2, token3, token4".A track match string is used when attaching a clip. It is compared against user data in the clip to select which tracks from the clip to map to player channels.

The string has the following format: token1/token2/token3,token4/token5 Matching occurs between a token and clip track user data value for key "ovn_trackmatch". See *OmPlrClipSetStartTimecode* and *OmPlrClipSetTrackUserData*. For the above track match string, the system first tries to find a clip track labeled token1, token2 or token3 in that order. The first match will be used. Then it will look for a clip track labeled token4 or token5. Again, the first match will be used. There are some special token names. The special name silenceX where X is a number will cause X channels of silence to be used. For example: a match string such as "english/silence2" will first look for a clip track labeled "english" and failing that will use 2 channels of silence. The special name **tracknumX** where X is a number will always match clip track number X.

The track match string is composed of UTF-8 characters. The typical tokens just use ASCII characters in the range of 0x20 to x07F; these typical characters happen to be valid UTF-8. The tokens in this string must match the tokens that are written into the track user data of the clip. The Spectrum SystemManager uses UTF-8 encoding in the TrackKeyValueDefs.txt file that it creates; this file is the source of the track tag tokens. An application that uses UTF-16 for most internal strings must be careful to use UTF-8 characters for this string.

The track match string pointer may be 0 or the track match string may be zero length.

See also

OmPlrAttach, *OmPlrOpen*, *OmPlrClipSetStartTimecode*, *OmPlrClipSetTrackUserData*, *Clip Name as Function Argument*

About pSubtitleFileList

pSubtitleFileList is a JSON string that may be used to specify how to override or replace parts of the caption configuration that were specified for the player.

The string contains a JSON array of JSON objects. Each JSON object corresponds to an association set in the player configuration, in the order specified in the configuration.

As an example, if the configuration specifies the following three entries:

1. cc1, cs1, tags="_en", file type="scc", extensions=".scc,.SCC",
2. cc3, cs2, tags="_sp", file type="scc", extensions=".scc,.SCC",
3. cs3, tags="_fr", file type="scc", extensions=".scc,.SCC"

The following JSON string would override the second entry:

```
[
  { },
  {
    'file' : "spanishCaps.Scc"
  }
]
```

The above JSON could be encoded in C/C++ as:

```
    const char *capConfig = "[{ }, {\"file\" :
\"spanishCaps.Scc\"}]";
```

or as:

```
    const char *capConfig = "[{ },{'file' : 'spanishCaps.Scc'}]";
```

The double quotes from the JSON need to escaped. However, single quotes do not need to be escaped, and thus might make the string easier to read in logs. Although single quotes are not part of the JSON specification, Spectrum accepts either single or double quotes for this input.

The first empty object, "{}" indicates that the player configuration settings should be used to select a file to be inserted for cc1/cs1.

The second object specifies that cc3/cs2 should have content from the file "spanishCaps.Scc". There is no third entry, so the cs3 entry will also use the settings from the player configuration.

Specifying an empty file string, "{ 'file' : '' }" can also be used to indicate the player configuration settings should be used to select a file. The following will also only modify the second entry.

```
[
  { 'file' : '' },
  {
    'file' : "spanishCaps.Scc"
  }
]
```

If there are more JSON objects in the array than configuration entries, the extra items will be ignored. Note that, in this case, Spectrum may generate a warning. If there are fewer JSON objects in the array than configuration entries, the missing items will be treated as empty, and configuration values will be used.

If the configured subtitle directory is "subtitle.dir," the directory

"/fs0/clip.dir/2016_02_14_21_00" has the following file and subdirectory:

```
myClip.mxf
subtitle.dir
```

and the "/fs0/clip.dir/2016_02_14_21_00/subtitle.dir" subdirectory has the following files:

```
myClip_en.scc
myClip_fr.scc
myClip_sp.scc
spanishCaps.Scc
opnCaps.stl
```

then:

- cc1/cs1 will have closed captions from the file myClip_en.scc

  The first object in the JSON array is empty, so the configuration is used to select the insertion file.

- cc3/cs2 will have closed captions from the file spanishCaps.Scc

  The second object in the JSON array has values which override the corresponding values in the configuration.

- cc3/cs3 will have closed captions from the file myClip_fr.scc

  There is no third object in the JSON array, so it is treated as an empty object and the third configuration settings are used to select the file.

If open captions are to be specified, the object file property name should be "opencap_file," as in:

```
[
   {
     'opencap_file' : 'opnCaps.stl'
   }
 ]
```

Harmonic recommends that the open captions object be put at the top of the JSON array. It will be accepted anywhere in newer systems (for Spectrum 8.2 and later).

Additional open caption field names are also supported in the open captions object. These are:

- **opencap_layer**. This specifies a layer in the range 1-4 or 1-8 (depending on license types).
- **opencap_template**. This specifies the name of the template to use.

These are optional values. If omitted, the values in the configuration will be used. For example, the JSON configuration that specifies both closed caption files and open captions might look like the following:

```
[
   {
     'opencap_file' : 'opnCaps.stl',
     'opencap_layer' : '6',
     'opencap_template' : '1080i60-captions-latin1.swf'
   },
   {
     'file' : 'englishCC.Scc'
   },
   {
     'file' : 'spanishCC.Scc'
   }
 ]
```

This JSON array has three objects. The first object is the open captions configuration, and the last two are closed caption objects that correspond to the first two configuration entries. Note that the lack of a third closed caption object in the JSON array indicates that the configuration settings should be used.

This previous JSON array might be encoded in a string as:

```
const char *exampleString =
    "[ { }, { 'opencap_file' : 'opnCaps.stl', "
    "'opencap_layer' : '6',"
```

```
"'opencap_template' : '1080i60-captions-latin1.swf' }, "
" { 'file' : 'spanishCaps.Scc' } ]";
```

The following JSON structure is an example teletext configuration that would work with a player configuration that supports insertion of three or more teletext subtitle configurations (for example, for OP-47), and an open captions entry:

```
[
    {
      'opencap_file' : 'opnCaps.stl',
      'opencap_layer' : '6',
      'opencap_template' : '1080i60-captions-latin1.swf'
    },
    {
      'file' : 'german.stl'
    },
    {
      'file' : 'english.stl'
    },
    {
      'file' : 'spanish.stl'
    }
  ]
```

Note this example includes an open captions configuration section followed by three files that correspond to the first three languages specified in the SystemManager configuration.

Be aware that loading an open captions template can take two seconds or more depending on the size of font set used in the template. Chinese, for example, has large fonts. As a result, multiple Chinese fonts may require several seconds to load.

If a single open captions template can be used for all video clips, then the load time will be incurred once for the first clip. The template will not be reloaded unless the template changes. If more than one open captions objects is found in the list, the first one will be used and the others will be discarded.

OmPlrDetach

Description

Removes an individual clip from the timeline.

```
OmPlrError OmPlrDetach(
OmPlrHandle playerHandle,
OmPlrClipHandle omPlrClipHandle,
OmPlrShiftMode shift);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
omPlrClipHandle	A numeric value that is used as a"handle," to identify the clip as it is attached to the timeline. The "handle" was obtained from the function **OmPlrAttach** or a function like **OmPlrGetClipAtNum**. The magic values of **omPlrFirstClip** and **omPlrLastClip** can also be used.
shift	When a clip is removed from the middle of a set of already attached clips, a hole is created that will be automatically filled by changing some existing clips' attachment points. The shift parameter is used to indicate how you want the hole to be filled. Possible values are: **omPlrShiftModeAfter**, **omPlrShiftModeBefore**, and **omPlrShiftModeAuto**. The value of **omPlrShiftModeOthers** doesn't apply when detaching clips, and should not be used. The value of **omPlrShiftModeAfter** causes the hole to be filled by moving clips **after** the removed clip so that these clips will be shown sooner. The value of **omPlrShiftModeBefore** causes the hole to be filled by shifting the previously attached clips before the removed clip to later timeline locations. A value of **omPlrShiftModeAuto** uses one of the two modes of **omPlrShiftModeAfter** or **omPlrShiftModeBefore**. The choice will be made to avoid changing the media at the **current position**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This function will be refused while the Player state is any form of recording. In that case, you will need to first use the **OmPlrStop** function. This function does not change the motion state or Player rate. If the Player was in the **omPlrPlay** or **omPlrCueplay** states, the Player continues to broadcast frames onto the Spectrum 1394 network; if there are no remaining clips attached, the Player will be sending black frames. The "number" assigned to other clips attached to the timeline may change after this function has executed. The values **omPlrShiftModeBefore**, **omPlrShiftModeAfter**, and **omPlrShiftModeAuto** are part of an enumeration that is defined in the file omplrdefs.h.

See also

Timeline Function Discussion, *Clip Function Discussion OmPlrAttach*, *OmPlrDetachAllClips*, *OmPlrOpen*

OmPlrDetachAllClips

Description

Removes all clips from the timeline

```
OmPlrError OmPlrDetachAllClips(
OmPlrHandle playerHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This function can be used while the Player state is recording; this is a change from previous behavior of the Spectrum MediaDirector. If you detach the clip immediately after the last frames were recorded into the particular clip, you may lose some frames from the clip; to avoid losing the frames either first issue a **OmPlrStop** command or wait several seconds. The stop command has the property of waiting for all steps of recording to complete.

CAUTION: This function does not change the motion state or rate of the Player. If the motion had been play at a speed of 32X, then it will still be that speed when the next clip is loaded. A **OmPlrStop** function will insure that the next loaded clip not instantly take out playing. Because this function does not change the motion state of the Player, a Player that was in the **omPlrPlay** or **omPlrCueplay** states will be broadcasting black frames onto the Spectrum 1394 network. The Player that had been in the omPlrStop state will not be sending any media onto the Spectrum network. In the typical case of using the Spectrum MediaPorts, if you want a defined output signal state after using this function, then you will need to combine this function with the **OmPlrStop** function or the **OmPlrCuePlay** function.

See also

Timeline Function Discussion, *Clip Function Discussion*, *OmPlrAttach*, *OmPlrOpen*

OmPlrGetClipAtNum

Description

Used to obtain a handle for a clip on the timeline.

```
OmPlrError OmPlrGetClipAtNum(
OmPlrHandle playerHandle,
int num,
OmPlrClipHandle *pOmPlrClipHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
num	A numeric value that identifies the desired clip in terms of its position on the timeline. The first clip on the timeline (i.e., the earliest) is assigned a num value of 0.
pOmPlrClipHandle	A required pointer used to return the "handle" for the clip. The "handle" is an arbitrary number that will be needed as a parameter for future calls that alter this clip on the timeline or that remove it from the timeline. Note: a valid "handle" will never have a value of 00.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

If the **num** parameter is the same as (or larger than) the actual number of clips attached to the timeline, then this function will return OK and will return an invalid handle value of 00.

The number assigned to each clip on the timeline may change when clips are deleted or attached.

See also

Clip Function Discussion, *Timeline Function Discussion*

OmPlrGetClipAtPos

Description

Used to obtain a handle for a clip on the timeline.

```
OmPlrError OmPlrGetClipAtPos(
OmPlrHandle playerHandle,
int timelinePosition,
OmPlrClipHandle *pOmPlrClipHandle,
int *pClipStartPos);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
timelinePosition	A numeric value that identifies the desired clip in terms of a timeline position. The value will be the timeline position expressed as binary frames. The timeline typically starts at a value of 00 but can start at negative values.
pOmPlrClipHandle	A pointer used to return the "handle" for the clip. Use a value of 00 for this argument if you do not want tho have the **OmPlrClipHandle** value returned. The "handle" is an arbitrary number that will be needed as a parameter for future calls that alter this clip on the timeline or that remove it from the timeline. Note: a valid "handle" will never have a value of 00.
pClipStartPos	A pointer used to return the starting point of the identified clip on the timeline (i.e., the "attachment" point of the clip on the timeline). Use a value of 00 for this argument if you do not want to have the **ClipStartPos** value returned.

Return value

If there is no clip at the specified timeline position, this function returns "all ok" (0), returns 0 as the value of **PlrClipHandle**, and returns a garbage number for **ClipStartPos**.

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

If the **timelinePosition** parameter is beyond the end of the last clip of the timeline or less than the start of the timeline, then this function will return OK and will return a handle value of 00, which is invalid; the returned value of **ClipStartPos** will be garbage in this case. The timeline parameters **minPos** and **maxPos** do not have any influence on the action of this function.

See also

Clip Function Discussion, *Timeline Function Discussion*

OmPlrGetClipData

Description

Used to return information about a clip as it has been mounted on the timeline.

```
OmPlrError OmPlrGetClipData(
OmPlrHandle playerHandle,
OmPlrClipHandle omPlrClipHandle,
int *pClipAttachPoint,
uint pClipIn,
uint *pClipOut,
uint *pClipLen);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
omPlrClipHandle	A numeric value that is used as a"handle," to identify the clip as it is attached to the timeline. The "handle" was obtained from the function **OmPlrAttach** or a function like **OmPlrGetClipAtNum**. The magic values of **omPlrFirstClip** and **omPlrLastClip** can also be used.
pClipAttachPoint	A pointer used to return the timeline "attachment point" value for this clip. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.
pClipIn	A pointer used to return the frame number of the first frame of this clip to be shown. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.
pClipOut	A pointer used to return the frame number of the frame after the last frame of this clip to be shown. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.
pClipLen	A pointer used to return the total number of timeline frames that will show this clip. Note that this value will not be necessarily be the same as **ClipOut** - **ClipIn**. For instance, interlace clips on a progressive player timeline (for example, a 25 Hz clip on a 50 Hz player) will have a timeline duration (**ClipLen**) that is twice the attached clip length (**ClipOut** - **ClipIn**). If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The constants **omPlrFirstClip** and **omPlrLastClip** are defined in the file omplrdefs.h; current values are 1 for **omPlrFirstClip** and 2 for **omPlrLastClip**.

The values returned for **clipIn** and **clipOut** are the frame numbers of the recorded clip and are related to the **firstFrame** value found in the clip information for this clip. The returned **clipIn** and **clipOut** values are not related to the timeline position values where this clip is attached to the timeline.

The values of **clipIn** and **clipOut** are tied to the frame numbering of the clip; the values are not timeline numbers.

See also

Clip Function Discussion, *Timeline Function Discussion*, *Player Function Discussion*, *OmPlrAttach*

OmPlrGetClipData1

Description

Another version of OmPlrGetClipData, this is used to return information about a clip on the timeline. This one adds a video frame conversion, active format description, and secondary audio mix select. This can be modified with OmPlrSetClipData() and OmPlrSetClipVfc().

```
OmPlrError OmPlrGetClipData1(
OmPlrHandle plrHandle,
OmPlrClipHandle clipHandle,
int *pPos,
uint pClipIn,
uint *pClipOut,
uint *pClipLen,
OmPlrVideoFrameConvert *vfc,
uint *afd,
uint *secAudMixSel);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
omPlrClipHandle	A numeric value that is used as a"handle," to identify the clip as it is attached to the timeline. The "handle" was obtained from the function **OmPlrAttach** or a function like **OmPlrGetClipAtNum**. The magic values of **omPlrFirstClip** and **omPlrLastClip** can also be used.
pPos	A pointer used to return the timeline position of the start of clip.
pClipIn	A pointer used to return the frame number of the first frame of this clip to be shown. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.
pClipOut	A pointer used to return the frame number of the frame after the last frame of this clip to be shown. If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.
pClipLen	A pointer used to return the total number of timeline frames that will show this clip. Note that this value will not be necessarily be the same as **ClipOut** - **ClipIn**. For instance, interlace clips on a progressive player timeline (for example, a 25 Hz clip on a 50 Hz player) will have a timeline duration (**ClipLen**) that is twice the attached clip length (**ClipOut** - **ClipIn**). If you do not want to have this value returned, then pass 00 for the pointer. If the function returns an error, nothing will be written using the pointer.
OmPlrVideoFormatConvert *vfc	The video frame conversion mode as specified in omPlrAttach2().

Parameter	Description
afd	The clip afd as specified in omPlrAttach3().
secAudMixSel	The clip secondary audio mix selection as specified in omPlrAttach3().

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The constants **omPlrFirstClip** and **omPlrLastClip** are defined in the file omplrdefs.h; current values are 1 for **omPlrFirstClip** and 2 for **omPlrLastClip**.

The values returned for **clipIn** and **clipOut** are the frame numbers of the recorded clip and are related to the **firstFrame** value found in the clip information for this clip. The returned **clipIn** and **clipOut** values are not related to the timeline position values where this clip is attached to the timeline.

The values of **clipIn** and **clipOut** are tied to the frame numbering of the clip; the values are not timeline numbers.

See also

Clip Function Discussion, *Timeline Function Discussion*, *Player Function Discussion*, *OmPlrAttach*

OmPlrGetClipName

Description

Gets the name of a clip that has been attached to the timeline.

```
OmPlrError OmPlrGetClipName(
OmPlrHandle playerHandle,
OmPlrClipHandle omPlrClipHandle,
TCHAR *pClipName,
uint clipNameSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
omPlrClipHandle	A numeric value that was assigned by the MediaDirector to identify the clip as attached to the timeline. The "handle" was obtained from the function **OmPlrAttach** or a function like **OmPlrGetClipAtNum**. The magic values of omPlrFirstClip and omPlrLastClip can also be used.

Parameter	Description
pClipName	A pointer to a buffer that will be used for returning the **ClipName**. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. In the case of any error, nothing will be written to the buffer.
clipNameSize	Gives the size of the buffer that **pClipName** points at. If the buffer is smaller than length of the clip name string (inclusive of the terminating '\0'), then the function will fail and will return an error of **omPlrBufferTooSmall**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

If there was an error returned, then nothing will be written to the "**ClipName**" buffer.

The constants **omPlrFirstClip** and **omPlrLastClip** are defined in the file omplrdefs.h; current values are 1 for **omPlrFirstClip** and 2 for **omPlrLastClip**.

See also

Clip Function Discussion, *Timeline Function Discussion*, *OmPlrAttach*

OmPlrGetClipFullName

Description

Returns full name information about attached clips.

```
OmPlrError OmPlrGetClipFullName(
OmPlrHandle plrHandle,
OmPlrClipHandle clipHandle,
TCHAR *pClipName,
uint clipNameSize);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
clipHandle	A numeric value used to reference the clip.
pClipName	A pointer to a buffer that will be used for returning the **ClipName**. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. In the case of any error, nothing will be written to the buffer.
clipNameSize	A numeric value that dictates the length of the clip name.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This function sets the **clipNameSize** to the size (in bytes) of the returned **ClipName**.

See also

OmPlrOpen, *Clip Name as Function Argument*

OmPlrGetClipPath

Description

Finds out the pathname for a clip that has already been attached to the timeline.

```
OmPlrError OmPlrGetClipPath(
OmPlrHandle playerHandle,
OmPlrClipHandle omPlrClipHandle,
TCHAR *pClipPath,
uint clipPathSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
omPlrClipHandle	A numeric value "handle", that identifies the clip as it is attached to the timeline. The "handle" was obtained from the function **OmPlrAttach** or a function like **OmPlrGetClipAtNum**. The magic values of **omPlrFirstClip** and **omPlrLastClip** can also be used.
pClipPath	A pointer to a buffer that will be used for returning a clip directory string. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. It will be written into the buffer as a "proper" C style string that has a byte of value 00 as a terminator. In the case of any error, nothing will be written to the buffer. The string gives the path that was used when this clip was attached to the timeline.
clipPathSize	Gives the size of the buffer that **pClipPath** points at. If the buffer is smaller than length of the clip path name string (inclusive of the terminating '\0'), then the function will fail and will return an error of **omPlrBufferTooSmall**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

*Remarks *

The returned path name will be an ABSOLUTE path name. The path was the current clip directory at the time that the **OmPlrAttach** function was used for this clip. The constant **omPlrMaxClipDirLen** is defined in the file omplrdefs.h; its current value is 512, which includes the terminating '\0' character. The constants **omPlrFirstClip** and **omPlrLastClip** are also defined in the file omplrdefs.h; current values are 1 for **omPlrFirstClip** and 2 for **omPlrLastClip**.

See also

Clip Function Discussion, *OmPlrClipGetDirectory*, *OmPlrAttach*

OmPlrGetTcgInsertion

Description

Gets the insertion settings of the internal timecode generator.

```
OmPlrError OmPlrGetTcgInsert(
OmPlrHandle playerHandle,
bool *pPlayInsert,
bool *pRecordInsert);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
pPlayInsert	A pointer used to return the current playback insert mode for timecode that was set by **OmPlrSetTcgInsertion**.
pRecordInsert	A pointer used to return the current record insert mode for timecode that was set by **OmPlrSetTcgInsertion**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

See the *Timecode Discussion* for more detail and for examples.

See also

OmPlrSetTcgInsertion, *Timecode Discussion*

OmPlrGetTcgMode

Description

Gets the mode for the internal timecode generator.

```
OmPlrError OmPlrGetTcgMode(
OmPlrHandle playerHandle,
OmPlrTcgMode *pMode);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
pMode	A pointer used to return the current timecode generator mode that was set by **OmPlrSetTcgMode**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

See also

OmPlrSetTcgMode, *Timecode Discussion*

OmPlrLoop

Description

Sets the first and last timeline positions to establish a playback looping mode.

```
OmPlrError OmPlrSetLoop(
OmPlrHandle playerHandle,
int minLoopPos,
int maxLoopPos);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
minLoopPos	A numeric value that establishes the timeline position for the first frame that will be shown during looping operation.
maxLoopPos	A numeric value that establishes the timeline position for the last frame during looping operation. The frame at this position will not be shown during the loop; the last frame seen will be at **maxLoopPos**–1. Looping operation is disabled if **maxLoopPos** is set to the same value as **minLoopPos**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The **minLoopPos** and **maxLoopPos** numbers are typically positive but can legitimately be negative values. There is no checking on the values used for setting **minLoopPos** or **maxLoopPos**. All values are valid. If the values select a range that is beyond the limits of the clips attached on the timeline, then black will be shown for those parts that are beyond the limits of attached clips. Notice that **minLoopPos** and **maxLoopPos** are timeline positions rather than being related to a particular clip.

Looping is disabled by setting **minLoopPos** to the same value as **maxLoopPos**. Looping is disabled initially when the Player was created in the MediaDirector. **OmPlrCueRecord** sets **minLoopPos** and **maxLoopPos** to 00, disabling any previously existing loop mode. Looping operation is inhibited during record. The looping mode is NOT cleared when all clips are detached.

When looping mode is established by setting the **minLoopPos** and the **maxLoopPos**, the position will jump if it had been outside the new looping limits; there is no jump if the position was already within the looping bounds. The jump is a result of some modulo math. Whenever looping is disabled, the Player's **MinPos** and **MaxPos** once again become effective and may force a jump of position.

Looping operation will be started when the next Play function is given. Looping operation can be at any speed in any direction. The timeline parameters **MinPos** and **MaxPos** are ignored while looping is enabled. Looping can also be enabled while already in play – but you will need to pay special attention to the current value of timeline position and may want to set the position if it is outside the new loop. Looping can be disabled while in play.

Changing the clips on the timeline while looping is also possible – but will be a full test of your timeline understanding. Attaching clips, detaching clips, or changing in/out points for clips that have already been attached may shift the timeline relative to your looping limits. You may need to adjust the looping limits. And the timeline position may have jumped when the timeline was changed if the shifted position was outside the looping limits.

If the **minLoopPos** value is greater than maxLoopPos, then the looping operation will execute in a bounce fashion, repeatedly playing from min to max and then playing in the inverse speed from max to min. Bounce mode is unsupported; you should always have the **maxLoopPos** value be the same or larger than the **minLoopPos**.

See also

Timeline Function Discussion, *OmPlrGetPlayerStatus*

OmPlrSetClipData

Description

Used to change properties of a clip as attached to the timeline.

```
OmPlrError OmPlrSetClipData(
OmPlrHandle playerHandle,
OmPlrClipHandle omPlrClipHandle,
uint clipIn,
uint clipOut,
uint clipLen,
OmPlrShiftMode shift);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
omPlrClipHandle	A numeric value that identifies the clip as it is attached to the timeline. The **omPlrClipHandle** was obtained from the function **OmPlrAttach** or a function like **OmPlrGetClipAtNum**. The magic values of **omPlrFirstClip** and **omPlrLastClip** can also be used.
clipIn	A numeric value that is the first frame of this clip to be shown. The special value **omPlrClipDefaultIn** can be used; it causes the defaultIn frame of the clip to be the first frame shown.
clipOut	A numeric value that is the frame **after** the last frame of this clip to be shown. The special value **omPlrClipDefaultOut** can be used; it causes the **defaultOut** frame value of the clip to be used for the **clipOut** value. An error will be returned if the **clipOut** value is less than the **clipIn** value.
clipLen	The special value **omPlrClipDefaultLen** should always be used for this argument. **omPlrClipDefaultLen** means that the result of calculating "Out – In" will be used for the **clipLen** parameter (the calculation is made after any needed evaluating of the magic values of **omPlrClipDefaultIn** or **omPlrClipDefaultOut**).
shift	A numeric value of **omPlrShiftModeBefore**, **omPlrShiftModeAfter**, **omPlrShftModeOthers**, or **omPlrShiftModeAuto**; these are defined in the file omplrdefs.h. When an attached clip is changed in the middle of a set of already attached clips, then either the clips before or after the changed clip will need to be shifted on the timeline to remove any gaps or overlaps. In some cases the changed clip will also need to be moved. The **shift** parameter is used to indicate your desire on how the clips should be rearranged. The value of **omPlrShiftModeAfter** causes the attachment points of the clips after "this" clip to be changed. The value of **omPlrShiftModeBefore** causes the attachment points of "this" clip and the clips attached before it to be changed. The value of **omPlrShiftModeOthers** causes the clips before and after this clip to be shifted such that this clip doesn't shift on the timeline; for instance, if frame 150 of this clip was attached before and was still attached after the **SetClipData** command, then that frame 150 will be at the same timeline position afterwards that it had before. A value of **omPlrShiftModeAuto** uses one of the three modes of **omPlrShiftModeAfter** or **omPlrShiftModeBefore** or **omPlrShiftOthers**; the choice will be made to avoid changing the media at the "current position."

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This function only changes the clip properties as it is attached to the timeline. This function does **NOT** change the default IN and OUT values that are stored with the clip on the MediaStore; the function that changes the points stored on the file system is **OmPlrClipSetDefaultInOut**.

The values used for **clipIn** and **clipOut** are the frame numbers of the recorded clip; for instance, to view the very first recorded frame of the clip, the **clipIn** value would be the **firstFrame** value found in the clip information for this clip. The **clipIn** and **clipOut** values are not related to the timeline position values where this clip is attached to the timeline.

The values of **clipIn** and **clipOut** are tied to the frame numbering of the clip; the values are not timeline numbers.

clipIn and **clipOut** can define a clip that starts or ends outside of the recorded video. You get black output for the un-recorded parts of the clip. This function can also be used to alter the **clipIn** and **clipOut** for empty clips that are being prepared for recording. In the case of an empty clip, **clipOut** – **clipIn** will determine how much material gets recorded into the clip.

omPlrShiftModeBefore can cause the timeline to start at a negative value. This is valid (timeline position is a signed number) but typically the timeline starts at a value of 00.

NOTE: Changing a clip on the timeline does NOT alter the maximum and minimum points for the timeline. Typically, you will also want to alter these values so that the timeline is allowed to march across the clip and show it to you.

The values **omPlrShiftModeBefore**, **omPlrShiftModeAfter**, **omPlrShiftModeOthers**, and **omPlrShiftModeAuto** are part of an enumeration that is defined in the file omplrdefs.h. At this time, the values are:

```
enum OmPlrShiftMode {
      omPlrShiftModeAfter,   // shift clips after the modified
   clip
      omPlrShiftModeBefore,  // shift clips before the modified
   clip
      omPlrShiftModeAuto,     // shift as necessary to avoid
   changing the current position
      omPlrShiftModeOthers// shift all other clips, not this one
};
```

The constants **omPlrFirstClip** and **omPlrLastClip** are defined in the file omplrdefs.h; current values are 1 for **omPlrFirstClip** and 2 for **omPlrLastClip**.

See also

Player Function Discussion, *Clip Function Discussion*, *Timeline Function Discussion*, *OmPlrSetMaxPos*, *OmPlrSetMinMaxPosToClip*, *OmPlrRecord*, *OmPlrPlay*

OmPlrSetMaxPos

Description

Sets the Maximum allowed position on the timeline.

```
OmPlrError OmPlrSetMaxPos(
OmPlrHandle playerHandle,
int maxPos);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
maxPos	The new value for the Maximum position on the timeline.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The **maxPos** position sets the upper limit for the timeline position value. The frame at **maxPos** position will not be shown. During play operation, the output will freeze on the previous frame and will repeatedly show that frame. During recording, the recording process will have been suspended after the frame before **maxPos** had been recorded into the new clip. Note that the value of **maxPos** does not matter during loop operations (but it isn't changed by loop operations).

The value of **maxPos** can be any value. No error condition is returned if it is set to less than the current setting of **minPos**. The value of **minPos** is not changed when the function is used. If the new value of **maxPos** is equal or less than the current timeline position, then the pos will be set to 1 less than the new **maxPos** value. If the new value of **maxPos** is equal or less than the value of **minPos**, then the current timeline position will be set to 00 and no change of position will be allowed.

The **maxPos** value is a timeline position number; this value is not tied to clip frame numbers.

CAUTION: If setting both timeline position and using this function to set the timeline's **maxPos** parameter, call this function before setting the timeline position.

See also

Timeline Function Discussion, *OmPlrSetMinMaxPosToClip*

OmPlrSetMaxPosMax

Sets the Maximum allowed position on the timeline at 1 beyond the end of last clip.

```
OmPlrError OmPlrSetMaxPosMax(
OmPlrHandle playerHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This function sets the Max Position value to 1 more than the timeline position of the last frame of the last clip. If there are no clips attached, then the **maxPos** value will be set to 00.

The Max Position value sets the upper limit for the timeline position value. The frame at **maxPos** position will not be shown. During play operation, the output will freeze on the previous frame and will repeatedly show that frame. During recording, the recording process will have been suspended after the frame before **maxPos** had been recorded into the new clip. Note that the value of Max Position does not matter during loop operations (but it isn't changed by loop operations).

Note that this Max Position value is related to the timeline numbers and not the OUT point of the last clip. The function **OmPlrSetMinMaxPosToClip** makes it easy to set **minPos** and **maxPos** in the case of only one clip being attached to the timeline.

See also

Timeline Function Discussion, *OmPlrSetMaxPos*, *OmPlrSetMinMaxPosToClip*

OmPlrSetMinMaxPosToClip

Description

Sets the Minimum and Maximum allowed positions on the timeline using the start and end of "this" clip.

```
OmPlrError OmPlrSetMinMaxPosToClip(
OmPlrHandle playerHandle,
OmPlrClipHandle omPlrClipHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
omPlrClipHandle	A numeric value that identifies the clip as it is attached to the timeline. This "handle" was obtained from the function **OmPlrAttach** or a function like **OmPlrGetClipAtNum**. The magic values of **omPlrFirstClip** and **omPlrLastClip** can also be used.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Sets the Minimum and Maximum allowed positions on the timeline using the start and end of "this" clip. A very useful function if there is only one clip attached to the timeline that you care about. Nothing happens if the 27clipHandle is not valid; the function returns OK in that case.

The **minPos** position sets the lower limit for the allowed values of the timeline position. The frame at **minPos** position will be seen if the timeline position is set to the minimum value. During play operation in a reverse direction, the output will freeze on this frame. The Max Position value sets the upper limit for the timeline position value. The frame at **maxPos** position will not be shown. During play operation, the output will freeze on the previous frame and will repeatedly show that frame. During recording, the recording process will have been suspended after the frame before **maxPos** had been recorded into the new clip. Note that the value of Min Position and Max Position do not matter during loop operations.

NOTE: Min and Max numbers are related to the timeline numbers. The Min number, for instance, will be set to the "attach point" of the clip rather than to the value of the clip's IN point.

The constants **omPlrFirstClip** and **omPlrLastClip** are also defined in the file omplrdefs.h; current values are 1 for **omPlrFirstClip** and 2 for **omPlrLastClip**.

See also

Timeline Function Discussion, *OmPlrSetMinMaxPosToClip*, *OmPlrSetMaxPos*

OmPlrSetMinPos

Description

Sets the Minimum position allowed on the timeline.

```
OmPlrError OmPlrSetMinPos(
OmPlrHandle playerHandle,
int minPos);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
minPos	A numeric value for the Minimum position on the timeline.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The **minPos** position sets the lower limit for the allowed values of the timeline position. The frame at **minPos** position will be seen if the timeline position is set to the minimum value. During play operation in a reverse direction, the output will freeze on this frame.

Note that the value of Min Position does not matter during loop operations (but it isn't changed by loop operations).

The value of **minPos** can be any value. No error condition is returned if it is set to equal or greater than the current setting of **maxPos**. The value of **maxPos** is not changed when the function is used. If the new value of **minPos** is greater than the current timeline position, then the **pos** will be set to the new **minPos** value. If the new value of **minPos** is equal or greater than the value of **maxPos**, then the current timeline position will be set to 00 and no change of position will be allowed.

CAUTION: If setting both timeline position and using this function to set the timeline's **maxPos** parameter, call this function before setting the timeline position.

See also

Timeline Function Discussion, *OmPlrSetMaxPos*, *OmPlrSetPos()*

OmPlrSetMinPosMin

Description

Sets the Minimum allowed position on the timeline to the first frame of the first clip.

```
OmPlrError OmPlrSetMinPosMin(
OmPlrHandle playerHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Sets the Minimum allowed position on the timeline to the first frame of the first clip. If there are no clips attached, then the **minPos** value will be set to 00.

The **minPos** position sets the lower limit for the allowed values of the timeline position. The frame at minPos position will be seen if the timeline position is set to the minimum value. During play operation in a reverse direction, the output will freeze on this frame.

Note that the value of Min Position does not matter during loop operations (but it isn't changed by loop operations).

Note that this Min Position value is related to the timeline numbers and not the IN point of the first clip. The function **OmPlrSetMinMaxPosToClip** makes it easy to set **minPos** and **maxPos** in the case of only one clip being attached to the timeline.

See also

Timeline Function Discussion, *OmPlrSetMinPos*, *OmPlrSetMinMaxPosToClip*

OmPlrSetTcgData

Description

Sets the data for the internal timecode generator.

```
OmPlrError OmPlrSetTcgData(
OmPlrHandle playerHandle,
OmTcData tcData,
bool onlyUserBits);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
tcData	Provides a new set of timecode and user bits for the timecode generator. In some cases only the user bits are used. The **OmTcData** structure is defined in the file OmPlrDefs.h
onlyUserBits	If true, only the user bit portion of the internal timecode generator is loaded with the data from the **tcData** parameter.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

What happens after this function is used depends on the current setting of **tcgMode**.

- If the mode is **omPlrTcgModeHold**, then the generator time will be exactly what you set.
- If the mode is **omPlrTcgModeFree**, then the generator time will start counting up from the value that was set.
- If the mode is **omPlrTcgModeLockedTimeline**, the generator time will be the setting value offset by the current timeline position.
- If the mode is **omPlrTcgModeLockedClip**, the generator time will be the setting value offset by the position inside the current clip.
- If the mode is **omPlrTcgModeLockedClipTc**, the generator time will be a fixed offset from a timecode value extrapolated from the clip initial timecode, where the clip initial timecode is the timecode value read from the first frame of the input media that was recorded.
- If the mode is **omPlrTcgModeLockedRefVitc**, the generator time is based on the reference VITC input.

See *Timecode Discussion* for more detail and for examples.

See also

OmPlrOpen

OmPlrSetTcgInsertion

Description

Enables insertion of the output of the internal timecode generator into video play or record streams.

```
OmPlrError OmPlrSetTcgInsert(
OmPlrHandle playerHandle,
bool playInsert,
bool recordInsert);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
playInsert	If this value is true, then timecode from the internal timecode generator will be inserted into the playback stream, replacing whatever time (and user bits) had originally been recorded with the clip.
recordInsert	If this value is true, then timecode from the internal timecode generator will be inserted into the record stream, replacing any time (and user bits) that is part of the incoming signal.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

See the *Timecode Discussion* section for more detail and for examples.

See also

Timecode Discussion, *OmPlrGetTcgInsertion*

OmPlrSetTcgMode

Description

Sets the mode for the internal timecode generator.

```
OmPlrError OmPlrSetTcgMode(
OmPlrHandle playerHandle,
OmPlrTcgMode mode);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
mode	The new value for the timecode generator mode; the mode is one of the values in the enumeration for **OmPlrTcgMode**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The definition of **OmPlrTcgMode** is an enumeration defined in the file omplrdefs.h. The current values of **OmPlrTcgMode** are:

```
        enum OmPlrTcgMode{
        omPlrTcgModeHold,
        omPlrTcgModeFreeRun,
        omPlrTcgModeLockedTimeline,
        omPlrTcgModeLockedClip,
        omPlrTcgModeLockedClipTc,
        omPlrTcgModeLockedRefVitc
};
```

Note the following regarding generator modes:

- **omPlrTcgModeHold** causes the generator data to remain fixed.
- **omPlrTcgModeFreeRun** causes the generator data to run continuously.
- **omPlrTcgModeLockedTimeline** causes the generator data to be a fixed offset from the current timeline position. This mode provides the accuracy needed for operations that need to match a specific timecode value to a specific recorded frame.
- **omPlrTcgModeLockedClip** causes the generator data to be a fixed offset from the current position within the current clip.
- **omPlrTcgModeLockedClipTc** causes the generator data to be a fixed offset from a timecode value extrapolated from the clip initial timecode where the clip initial timecode is the timecode value read from the first frame of the input media that was recorded. The clip timecode is read from either the LTC input of the MediaPort or from the Vertical Interval Timecode (VITC) that was embedded in the incoming video media. Selection of the timecode source is determined by the configuration of the player object. Extrapolation ensures that the generator timecode increases in a smooth monotonic fashion from its initial value even if the input timecode is discontinuous.
- **omPlrTcgModeLockedRefVitc** causes the generator data to be based on reference VITC input.

See the *Timecode Discussion* section for additional information and examples.

NOTE: The timecode generator mode does not automatically change at the end of a recording or other activity.

See also

Timecode Discussion, *OmPlrGetTcgMode*

OmPlrSetClipVfc

Description

Modified the video frame converion of a clip on the timeline.

```
OmPlrError OmPlrSetClipVfc(
OmPlrHandle plrHandle
OmPlrClipHandle clipHandle,
OmPlrVideoFrameCovert vfc);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
omPlrClipHandle	A numeric value that is used as a“handle,” to identify the clip as it is attached to the timeline. The “handle” was obtained from the function **OmPlrAttach** or a function like **OmPlrGetClipAtNum**. The magic values of **omPlrFirstClip** and **omPlrLastClip** can also be used.
OmPlrVideoFormatConvert *vfc	The video frame conversion mode as specified in omPlrAttach2().

Chapter 5
Player Discovery and Other Functions

This chapter provides Spectrum API Functions related to Player Discovery in addition to other functions that are not tied to a particular Player. The following functions are included:

Function	Description
OmPlrClose	Close a connection to a remote Player.
OmPlrEnable	Enable/disable Players in Spectrum video servers.
OmPlrGetApp	Discover the "controlling application" property of the Player.
OmPlrGetAppClipExtList	Get the current Player clip extension list.
OmPlrGetAppClipDirectory	Discover the "suggested clip directory" property of the Player.
OmPlrGetDropFrame	Get the current Player-configured preference for drop or non-drop time display mode.
OmPlrGetErrorString	Convert a "playerError_t"error number into a string.
OmPlrGetFirstPlayer	Get the name of the first Player in a list of players.
OmPlrGetNextPlayer	Get the name of the "next" Player in a list of players.
OmPlrGetPlayerName	Returns the name of the Player for "this" connection.
OmPlrGetFrameRate	Get the Player's frame rate.
OmPlrOpen	Establish a connection to a Player object on a particular MediaDirector.
OmPlrOpen1	Opens a connection to a Player on a MediaDirector.
OmPlrSetRetryOpen	Enable/disable library code from trying to reopen lost connections to the MediaDirector.
OmPlrSetPlayerSwitching	Enables automatic player switching.
OmPlrGetTime	Query system frame count and VITC reference time. See explanation in the **Player Status Functions** section.

OmPlrClose

Description

Closes a connection to a Player.

```
OmPlrError OmPlrClose(
OmPlrHandle playerHandle);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

This will close out the session using this **playerHandle** value. If the Player had been registered for clip change callbacks, this function will shut down the callback mechanism. No Player functions except for **OmPlrOpen** will succeed if used after this. Harmless if called with a **playerHandle** of 00 – except that the return value will be PLAYER_BAD_REMOTE_HANDLE.

See also

OmPlrOpen, *Player Function Discussion*

OmPlrEnable

Description

Allows applications to selectively enable or disable Players in Spectrum video servers. This allows automation to change the characteristics of the Players dynamically. For instance, it enables switching a MediaPort channel from SD to HD.

```
OMPLRLIB_C_API OmPlrError OmPlrEnable(
    OmPlrHandle plrHandle,
    bool enable);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
bool enable	Set to true to enable the Player or false to disable the Player.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Disabling a Player causes the Player to disallow attaching clips, cueing for play, or cueing for record. The Player must be stopped with no clips attached to disable a Player. Enabling a Player allows a set of players to exist where all but one is disabled. Automation can then select a player to use by disabling the currently active player and enabling the desired player.

The play or record capability determination has not changed. The Player API **OmPlrPlayEnabled()** returns a true value when the Player can play. This would be true if the Player was configured for play by SystemManager and the Player has not been disabled. The Player API **OmPlrRecordEnabled()** likewise reports record capability. The fields **playEnabled** and **recordEnabled** in the **OmPlrStatus** struct returned from the Player API **OmPlrGetPlayerStatus()** return these same values.

See also

OmPlrOpen, *Player Function Discussion*

OmPlrGetApp

Description

Discovers the "controlling application" property of the Player.

```
OmPlrError OmPlrGetApp(
OmPlrHandle playerHandle,
OmPlrApp *pPlayerApp);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pPlayerApp	A pointer used to return an enumerated value that indicates what controlling application has been selected for this Player. For instance, the value may indicate that this Player can be controlled by VDCP functions.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The current possible values for **PlayerApp** are defined as:

```
omPlrAppNone,
omPlrAppLouth,
omPlrAppOmnibus,
omPlrAppBvw,
omPlrAppAvc
```

The values are defined as an enumeration in the file omplrdefs.h.

The application type should be taken as a suggestion. No matter what has been chosen as the control application, the Player can also be controlled by the functions documented here. And the Spectrum ClipTool application is always able to control a Player.

Note that nothing is returned via the pointer if the function returns an error.

OmPlrGetAppClipExtList

Description

Gets the current player clip extension list.

```
OmPlrError OmPlrGetAppClipExtList(
OmPlrHandle plrHandle,
TCHAR *extList,
uint extListSize);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
extList	A pointer to a buffer that is used to store the returned extension list.
extListSize	A numeric value that shows the size in bytes of the **extList** buffer.

Return value

A numeric value that defines success or error. A value of 0 indicates “all ok.” Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Returns a string set which is a list of filename extensions.

Returns **OmPlrBadHandle** if called with a connection that was opened with a NULL player.

Remarks

The extension list is a concatenated list of filename extensions that are used as clip identifiers. For example: for quicktime files and dv files the list would look like ".mov.dv". This clip extension list is an attribute of the player, not an attribute of the network connection. This clip extension list is used for control from VDCP, AVC, etc. Set **extListSize** to the size (in bytes) of extList buffer

See also

OmPlrOpen

OmPlrGetAppClipDirectory

Description

Discovers the “suggested clip directory” property of the Player.

```
OmPlrError OmPlrGetAppClipDirectory(
OmPlrHandle playerHandle,
TCHAR *pClipDir,
uint clipDirSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pClipDir	A pointer to a buffer that will be used for returning the suggested clip directory. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. In the case of any error, nothing will be written to the buffer.

Parameter	Description
clipDirSize	Gives the size of the buffer that **pClipDir** points at. If the buffer is smaller than length of the clip directory name string (inclusive of the terminating '\0'), then the function will fail and will return an error of **omPlrBufferTooSmall**.

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Returns **OmPlrBadHandle** if called with a connection that was opened with a NULL player.

Remarks

The Spectrum SystemManager is used to create *Player* objects. One of the fields in the configuration form is the Default Clip Directory field. The **OmPlrGetAppClipDirectory** returns that entry. So this function provides a means for the application to determine the intentions of the person that configured the player object. Typically the application would then use the returned string as the argument for a **OmPlrClipSetDirectory** call. One key point is that the **AppClipDirectory** property doesn't directly effect the clip directory; it is just some information for you, the application writer, to make use of. The other point is that each player object has its own setting for the **AppClipDirectory** property.

Note that nothing is returned via the pointer if the function returns an error.

OmPlrGetDropFrame

Description

Gets the current Player-configured preference for drop or non-drop time display mode.

```
OmPlrError OmPlrGetDropFrame(
OmPlrHandle playerHandle,
bool *pDropFrameMode);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this player. It was obtained using the function *OmPlrOpen*.
pDropFrameMode	A pointer that is used to return a boolean value. The value will be true if drop frame mode has been selected as the system preference.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The drop frame mode returned by this function expresses someone's preference for how NTSC times should be displayed. The mode does not have any effect inside the Spectrum Player object because the Player only deals with times that are expressed as an absolute count of frames. There is only one drop frame preference for any particular Player. Different

players can have different settings. The individual clips do not carry a drop frame preference. Typically this information would be used by an application that wanted to convert a time from absolute frames back to a Hours/Minutes/Seconds/Frames format for the purpose of creating a user-friendly display. The drop frame preference is set when the Player is configured using the Spectrum SystemManager product.

Note that nothing is returned via the pointer if the function returns an error.

OmPlrGetErrorString

Description

Converts a "OmPlrError_t" error number into a string.

```
const TCHAR * OmPlrGetErrorString(
OmPlrError error);
```

Parameters

Parameter	Description
error	A numeric value that defines the error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors is 0x00009000 to 0x00009FFF.

Return value

Returns a pointer to a string. The string will be terminated with a '\0' byte. The string does NOT include a "new line."

Remarks

These are constant strings; they are part of the Spectrum provided client library and are not sent from the MediaDirector.

OmPlrGetFirstPlayer

Description

Returns the name of the first Player in a list of players.

```
OmPlrError OmPlrGetFirstPlayer(
OmPlrHandle playerHandle,
TCHAR *pPlayerName,
uint playerNameSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
pPlayerName	A pointer to a buffer that will be used for returning the name of the first Player. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. In the case of any error, nothing will be written to the buffer.
playerNameSize	Gives the size of the buffer that **pPlayerName** points at. If the buffer is smaller than length of the player name string (inclusive of the terminating '\0'), then the function will fail and will return an error of **omPlrBufferTooSmall**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." A value of **omPlrEndOfList** indicates that there are no players available. Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

Players are enumerated by first calling **OmPlrGetFirstPlayer** and then repeatedly calling **OmPlrGetNextPlayer**. Keep calling until the return value is **omPlrEndOfList**. The call that returns the value **omPlrEndOfList** will not return a name in the buffer. This function can be used even if a NULL Player name was used to open the connection.

Players are listed by the most recent creation.

A typical application would be to discover a Player that has the desired application. The connection would be established using **OmPlrOpen** with a NULL pointer for the Player name. Then a list of players would be gathered using this function and **OmPlrGetNextPlayer**. As each Player name is obtained, it would be used to open a second connection that connects to the Player and then a call would be made to obtain the application that had been assigned to this Player. The second connection would be closed before the start of the next iteration.

See also

OmPlrGetNextPlayer, *OmPlrOpen*

OmPlrGetNextPlayer

Description

Gets the name of the "next" Player in a list of players.

```
OmPlrError OmPlrGetNextPlayer(
OmPlrHandle playerHandle,
TCHAR *pPlayerName,
uint playerNameSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
pPlayerName	A pointer to a buffer that will be used for returning the name of the first Player. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. In the case of any error, nothing will be written to the buffer.
playerNameSize	Gives the size of the buffer that **pPlayerName** points at. If the buffer is smaller than length of the player name string (inclusive of the terminating '\0'), then the function will fail and will return an error of **omPlrBufferTooSmall**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." A value of **omPlrEndOfList** indicates that there are no players available. Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Players are listed by the most recent creation.

Remarks

Players are enumerated by first calling **OmPlrGetFirstPlayer** and then repeatedly calling **OmPlrGetNextPlayer**. Keep calling until the return value is **omPlrEndOfList**. The call that returns the value **omPlrEndOfList** will not return a name in the buffer. See *OmPlrGetFirstPlayer* for additional discussion.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrOpen, *OmPlrGetFirstPlayer*

OmPlrGetPlayerName

Description

Returns the name of the Player for "this" connection.

```
OmPlrError OmPlrGetPlayerName(
OmPlrHandle playerHandle,
TCHAR *pPlayerName,
uint playerNameSize);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
pPlayerName	A pointer to a buffer that will be used for returning the name of the first Player. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. In the case of any error, nothing will be written to the buffer.
playerNameSize	Gives the size of the buffer that **pPlayerName** points at. If the buffer is smaller than length of the player name string (inclusive of the terminating '\0'), then the function will fail and will return an error of **omPlrBufferTooSmall**.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

See also

OmPlrOpen, *OmPlrGetFirstPlayer*, *OmPlrGetNextPlayer*

OmPlrGetFrameRate

Description

Returns the player frame rate. The player frame rate does not need to match the reference frame rate.

```
OmPlrError OmPlrGetFrameRate(
    OmPlrHandle plrHandle,
    OmFrameRate *pFrameRate);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
pFrameRate	A pointer that will be used to return an enumerated value that indicates the frame rate of the player. Nothing will be returned via the pointer if the function returns an error condition.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

See also

OmPlrGetTime

OmPlrOpen

Description

Establishes a connection to a Player object on a particular MediaDirector.

```
OmPlrError OmPlrOpen(
const TCHAR * pDirectorName,
const TCHAR * pPlayerName,
OmPlrHandle * pPlayerHandle);
```

Parameters

Parameter	Description
pDirectorName	A required pointer to a string that is the MediaDirector name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. Maximum length of the string is **omPlrMaxDirectorNameLen**. The MediaDirector name can be either in the form of a domain name or a string that gives the "dotted IP address." Examples are "dirB38" and "10.35.80.208."
pPlayerName	An optional pointer to a string that is the Player name. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string. A value of 00 can be used as the pointer (i.e., a NULL pointer) if you only want general information from the MediaDirector and do not need to control a particular Player. Maximum length of the string is **omPlrMaxPlayerNameLen**.
pPlayerHandle	A pointer used to return a "handle" for this connection. An arbitrary number will be assigned to each connection. This number is returned to the caller to be used as a "handle". The "handle", will be needed as a parameter for all of the other function calls. **NOTE**: A valid "handle" will never have a value of 00.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The default connection is made using TCP/IP protocol over the ethernet link. If the MediaDirector name has a suffix of ":udp", then the ethernet connection will be made using UDP/IP protocol.

A NULL pointer for the **PlayerName** could be used if all you wanted was to list the clips on a MediaDirector. Or if you wanted to establish a connection and were then going to query for the set of available players on a MediaDirector. All of the OmPlrClip… functions will work with a NULL Player type of connection; many of the functions in this "Player discovery" category will also work. You need to connect to a specific Player in order to use the motion, timeline, and status function categories.

By default, connections are opened in “Retry Open” mode that will automatically try to re-establish connection to the MediaDirector if a future function call fails due to connection trouble. The automatic re-connect will also be useful if there is a long period of inactivity, as the MediaDirector closes any connection that has been idle for more than 8 minutes.

This function establishes a structure in the memory associated with the client library. The function **OmPlrClose** should be called when the connection is no longer needed. The **OmPlrClose** function will free the memory in the client space.

The constant **omPlrMaxDirectorNameLen** is defined in the file omplrdefs.h; its current value is 64, which includes the terminating ‘\0’ character.

When a MediaDirector starts up, the file system may not come up right away. In that case the rpc system will get started before the file system; API commands are supported once the rpc system has started. The player objects won’t be restored in the MediaDirector until after the file system is working and able to provide the player configuration information. BUT in the meanwhile you can establish a connection. During this interval between RPC system starting and file system starting, the commands that do not require a player will work (such as **OmPlrGetTime**) and the commands that do require a player will fail (such as **OmPlrGetPlayerStatus**). After the file system is running, the MediaDirector will start the players and will automatically complete the connection to the requested player, so that commands such as **OmPlrGetPlayerStatus** will start working without requiring a **Close** and an **Open**.

CAUTION: Immediately after a reset of a MediaDirector, it will be possible to connect to the MediaDirector before the MediaDirector has its Player objects ready. In the case of an “early” connect, the connection will have the same limited functionality as if it had been opened with a NULL Player name; the “early” connection remains limited even after the Player objects are ready. An “early connection” will need to be closed with **OmPlrClose** and then re-opened with this **OmPlrOpen** function if you need a connection to a specific Player.

In order to detect either of these cases, it is suggested that you test your connections after using this function. For instance, a successful call to a function such as **OmPlrGetPos** or **OmPlrGetPlayerStatus** using the returned handle will confirm that the connection is successfully tied to a Player object.

See also

Player Function Discussion, *OmPlrClose*, *OmPlrGetClipName*, *OmPlrSetRetryOpen*

OmPlrOpen1

Description

Opens a connection to a Player on a MediaDirector.

```
OmPlrError OmPlrOpen1(
    const TCHAR *pDirName,
    const TCHAR *pPlrName,
    OmPlrHandle *pPlrHandle,
    void (*pConnectCallback)(OmPlrHandle plrHand));
```

Parameters

Parameter	Description
pDirName	A pointer to the name of the MediaDirector (or dotted ip address) which contains the Player that should be opened. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string.
pPlrName	A pointer to the name of the Player that needs to be opened on the MediaDirector. For non-Unicode builds, this is an 8-bit character string. For Unicode builds, this is a 16-bit character string.
pPlrHandle	A pointer used to return a "handle" for this connection. An arbitrary number will be assigned to each connection. This number is returned to the caller to be used as a "handle". The "handle", will be needed as a parameter for all of the other function calls.
pConnectCallback	A pointer to the callback connection.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The **connectCallback** routine is called when a network connection to the MediaDirector is established or reestablished.

See also

OmPlrOpen

OmPlrSetPlayerSwitching

Description

Enables automatic player switching.

```
OmPlrError OmPlrSetPlayerSwitching(
OmPlrHandle plrHandle
bool enable);
```

Parameters

Parameter	Description
plrHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
bool enable	If true, automatic player switching is enabled.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

A connection to a player may automatically switch to another player that controls the same MediaPort. This player switch may occur when responding to OmPlrCuePlay, OmPlrCueRecord, OmPlrCueRecord1, OmPlrAttch, OmPlrAttach1 and OmPlrAttch2.

The connection switches to a player properly configured to record the current video input type when an intention to record is indicated by one of the above commands.

The connection switches back to the original player when an intention to play is indicated by one of the above commands.

Player switching is enabled by default. Use this command to disable player switching. This command should be called just after calling OmPlrOpen() when disabled player switching is desired.

OmPlrSetRetryOpen

Description

Enables/disables library code from trying to reopen lost connections to the MediaDirector.

```
OmPlrError OmPlrSetRetryOpen(
OmPlrHandle playerHandle,
bool retry);
```

Parameters

Parameter	Description
playerHandle	A numeric value used to reference this Player. It was obtained using the function *OmPlrOpen*.
retry	A boolean value, when set to "true," enables the retry action. By default, retrying is enabled by the function *OmPlrOpen*.

Return value

A numeric value that defines success or error. A value of 0 indicates "all ok." Error codes are defined in the file omplrdefs.h. The error codes are assigned in ranges. The range for Player errors starts at PLAYER_ERROR_BASE (0x00009000).

Remarks

The client library that supports all these functions established a connection to the MediaDirector when the function **OmPlrOpen** was used. If later the connection is broken, then there is code in the client library that will detect that a function call failed and will try to reopen the connection and then retry the function. The retry will be a maximum of once for each function call. The retry code can be enabled or disabled as a result of this call. The code was enabled by default for this **playerHandle** when the function **OmPlrOpen** was used.

This function can be used even if a NULL Player name was used to open the connection.

See also

OmPlrOpen

Appendix A
OPC Tester

This appendix provides the following information:

About the OPC Tester

This **opcTester** program is used for trying out the functions that are part of the Spectrum Player API. With this program, you can execute a series of functions just like you plan to use in your application that will control an Spectrum system. The **opcTester** program is a simple console based program that has a simple command line user interface.

Installing OPC Tester

On Windows

In the .\bin directory of the SDK distribution there are two files:

- opcTester.exe
- omPlrLib.dll

You can execute the **opcTester** application directing from that directory. Alternatively, copy those two files into another directory on your computer and then execute from this second directory.

On Linux

The **opcTester** application for Linux is distributed in source code form. To build it, cd to the ./src directory and type :

```
$ make
```

Once the application is built, you can execute it directly from that directory. Alternatively, copy the opctester binary into another directory on your computer and then execute from this second directory.

Starting the Program

The program can be started from a short cut or from a *command* window. Optionally it can accept the MediaDirector name or IP address as an argument. The MediaDirector name is obtained from the Spectrum SystemManager product. The MediaDirector name can optionally be followed by a player name. Example – "opcTester 10.35.80.208 player1." If opcTester is started without any arguments, then inside the program use the "Open" to make a connection to a MediaDirector.

Entering Commands

Commands are not case insensitive, meaning that they will be recognized if typed upper case, lower case, or a mixture of cases. You only need to type enough of the command to distinguish it from other commands. If several commands match the partial command that you entered, you will be shown a list of possible commands and will be given a chance to re-enter the command. For instance, "q" is enough to specify "quit" and "ClipSetP" will specify "ClipSetProtection."

- The QUIT command terminates the program. The word EXIT also works.
- The HELP command lists all the available commands. If you know the starting letter of the desired command, you can see a list of just the commands that start with that letter. For instance, if you know that the command in among the "get" commands, you can type "help g" to see all the commands that start with a "g."
- The HELP line for each command will give the command name and will give required and optional arguments and then gives a short description of the command. A argument is required is that is no bracketing around it. A argument is optional if it is bracketed with "[" and "]". Groups of arguments are separated by a "|" if you must pick one OR the other, but not more than one (or pick none if inside optional arguments). One of a group of arguments is required if the group is bracketed by "{" and "}"; e.g., "{ |A|+}" means that you have to use one of the three characters as an argument – but not more than one character.

Command Entry Details

opcTester is a typical Windows console program. So the backspace key works while entering commands. Also the UP arrow can be used to re-use previous commands. And the LEFT and RIGHT arrows can be used while editing the command. Overstrike or Insert mode is toggled with the INSERT key. Hit the ENTER key to execute the command.

Clip names and directory paths can have spaces embedded in them. The **opcTester** program supports the use of the double quote (") character to allow the entry if an embedded space. Use one double quote at the start of the clip name and one at the end of it. An example is – "this clip has spaces."

But then you have the problem of how to put the double quote into a clip name. To handle that problem, opcTester uses a back slash (\) as a second layer of *escape* character. The back slash is only special outside the limits of a quoted string. So a double quote is inserted into a clip name by breaking the name into two strings that are joined by an *escaped* quote. Examples are:

- "name"\""with embedded quote" -- in the middle, that is doubleQuote, backSlash, doubleQuote, doubleQuote.
- "name\with backslash" – a back slash inside a quoted string is just a plain back slash.
- Name\ space\ no\ quotes – embeds single spaces by preceding each with a back slash.

Anything after a '#' character on the command line is ignored. This allows you to type in comments for logging or for script files. The '#" is not a special character inside a quoted string. The '#' is also not a special character if preceded by a back slash.

Command Names and Types

The names of most of the commands are the same as the function names that are given in the API document except that the "OmPlr" prefix is omitted. For instance, the command to attach a clip to the timeline is ATTACH.

The **opcTester** puts more emphasis on matching the API document than on ease of use but it does have some features to make your life easier. The up and down arrows can be used to find previous commands for re-use. The previous commands can be edited using the typical Windows keystrokes. The command names can be long to type; take advantage of the feature that allows you to type in partial commands. Type in the first few letters of your desired command and hit enter to see if that was enough letters to uniquely define the command. If there weren't yet enough letters, then OpcTester will show you the set of commands that are possible matches for the letters that you typed; retype the command with more letters. If you are often repeating the same set of commands, consider putting them into a text file and then using the source command of OpcTester.

Example Commands

```
XBot - stop motion; go to Min timeline position
Xcreate clipName [in out] - Eject all and then create this clip <XC>
XcreateNext clipName [in out] - Append this clip to end of timeline
Xeject - stop motion; detach all clips
Xgoto  [clipNum] position - goto a position in a clip & pause
Xload clipName [in out] -Eject all and then load this clip <XL>
XloadNext clipName [in out] - Append this clip to end of timeline
Xplay [rate] [clipName] - Load if needed, then play the clip
Xrecord [clipName [numFrames ] - record a clip
Xshuttle [rate] - play at variable speed
Xstep  [stepSizeSign] - move X frames from current position
Xstill - show still frame
Xstop - stop motion; show input <XS>

XlistAttached  - list all clips attached to timeline <lt>
XlistClips  - list all clips in current directory <lc>
XlistPlayers  - list all players on current MediaServer <lp>  ***
```

These "X" style commands use multiple API commands to accomplish their goal. They exist for two reasons. (1) to make your life easier and (2) to give you example code for performing common tasks. For instance, the Xload command makes use of the API commands *OmPlrGetPlayerStatus*, *OmPlrClipGetInfo*, *OmPlrDetachAllClips*, *OmPlrAttach*, *OmPlrSetMinPosMin*, *OmPlrSetMaxPosMax*, and *OmPlrSetPos()*. The functions that support these "x" commands are named "Example_" and are in the file opcTestSamples.cpp.

opcTester Control Commands

QUIT

SOURCE

SLEEP

LINES

HELP

Support for Scripts

One of the **opcTester** commands is SOURCE. This command causes **opcTester** to access commands from a text file. The interactive mode is turned off while the commands are executed from the command text file – so the screen output will not be paused for any "continue yes/no" type of questions. After the last command of the command file has been executed, the program will prompt you for the next command. The command file needs to be in whatever directory on your pc that has been specified as the working directory of the **opcTester** program.

Or on startup, you can use the "-f" argument and specify the name of a command file. Same result as if you typed the SOURCE command.

Lastly, on startup the input of the **opcTester** program can be redirected so that it comes from "standard input" rather than from the Windows controlled keyboard. An argument of '-' tells **opcTester** of your intent; prompts and interactive queries about "continue" will be suppressed. An example of the command line is " opcTester - < someFile". In this case, be sure that the file has a QUIT command at the end of it.

Dictionary of Arguments

ClipNum — identifies a clip that has been attached to the timeline. The API functions require the **OmPlrClipHandle**; the **OmPlrClipHandle** is returned by the ATTACH command. Enter the clip handle as a hex number such as 0x6137258. But **opcTester** commands can also accept a clip NUMBER to make it easier for you (the human banging on the keyboard); enter the clip number as a decimal number such as "5." The clip number for the first clip attached is 0; the set of clip numbers is changed whenever a clip is attached or detached to the timeline. Remember, this is a short cut for your convenience; your code will need to remember the **OmPlrClipHandle** values in most cases if you want to later refer to the attached clip.

clipName — follows API document rules (so, for instance, case matters). When **opcTester** displays clip names, they are sometimes truncated so that other information fits on the same line. Truncation is an **opcTester** property and not an API property.

playerName — follows the rules of the API document so that case matters. Used with the Open command. Use a "." if you want to open using a NULL player (i.e., pass a pointer of 0 to **OmPlrOpen**).

directorName — use an IP address such as 10.35.74.200 or use a MediaDirector name. If you want to close one Player's connection on a MediaDirector and then open another connection on the same MediaDirector, use "." for the directorName argument; this works because opcTester remembers the name of the current MediaDirector.

clipCopyHandle — the **clipCopy** function returns a clip copy handle value. The opcTester code remembers this value and supplies it for functions such as **OmPlrClipCopyAbort**. This is handy but limits you to working with one copy at a time. Instead, supply the handle by entering it with the command, such as "clipCopyAbort 0x100003." To force opcTester to allow you to start multiple clip copys, use a **clipCopyX** command instead of **clipCopy**.

clipCopy first, length, firstFrame — the **OmPlrClipCopy** function has special values of ~0 for these parameters. You can enter either "0xFFFFFFFF" or "-1" into opcTester to ask for the special ~0 value.

Appendix B
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.

Report an issue online at:

http://harmonicinc.com/webform/report-issue-online

Table 0–1: Technical Support Phone Numbers and Email Addresses

Region	Telephone Technical Support	E-mail
Americas	888.673.4896 (888.MPEG.TWO) or +1.408.490.6477	*support@harmonicinc.com*
Europe, Middle East, and Africa	+44.1252.555.450	*emeasupport@harmonicinc.com*
India	+91.120.498.3199	*apacsupport@harmonicinc.com*
Russia	+7.495.926.4608	*rusupport@harmonicinc.com*
Mainland China	+86.10.6569.5580	*chinasupport@harmonicinc.com*
Japan	+81.3.5565.6737	*japansupport@harmonicinc.com*
Asia Pacific – Other Territories	+852.3184.0045 or 65.6542.0050	*apacsupport@harmonicinc.com*

The Harmonic Inc. support website is:

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

The Harmonic Inc. software download locations are:

All Harmonic software except Cable Edge software	Software updates are available from the Harmonic website. Contact Harmonic Technical Support for login information.
Cable Edge software	*ftp://ftp.harmonicinc.com*

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