grass valley

# Profile XP

## SOFTWARE DEVELOPMENT KIT

Reference Manual

SOFTWARE VERSION 5.1

SEPTEMBER 2003

*the most watched worldwide*

A THOMSON BRAND

## Copyright

Copyright © 2003 Thomson Broadcast and Media Solutions, Inc. All rights reserved. Printed in the United States of America.

This document may not be copied in whole or in part, or otherwise reproduced except as specifically permitted under U.S. copyright law, without the prior written consent of Thomson Broadcast and Media Solutions, Inc., P.O. Box 59900, Nevada City, California 95959-7900

## Trademarks

Grass Valley, Profile, and Profile XP are either registered trademarks or trademarks of Thomson Broadcast and Media Solutions, Inc. in the United States and/or other countries. Other trademarks used in this document are either registered trademarks or trademarks of the manufacturers or vendors of the associated products. Thomson Broadcast and Media Solutions, Inc. products are covered by U.S. and foreign patents, issued and pending. Additional information regarding Thomson Broadcast and Media Solutions, Inc. trademarks and other proprietary rights may be found at www.thomsongrassvalley.com.

## Disclaimer

Product options and specifications subject to change without notice. The information in this manual is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Thomson Broadcast and Media Solutions, Inc. Thomson Broadcast and Media Solutions, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this publication.

## U.S. Government Restricted Rights Legend

Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7013 or in subparagraph c(1) and (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19, as applicable. Manufacturer is Thomson Broadcast and Media Solutions, Inc., P.O. Box 59900, Nevada City, California 95959-7900 U.S.A.

## Revision Status

| Rev Date | Description |
|---|---|
| May 1996 | Original issue; manual part number 070-9643-00. |
| May 1999 | Updated to support version 3.0. Manual part number changed to 070-9643-01. |
| April 2000 | Updated to support version 4.0. Part number 071-8020-00. |
| June 2002 | Updated to support version 5.0. Preliminary. |
| November 2002 | Updated to support version 5.1. Preliminary. |
| September 24, 2003 | Corrected bitmask information for PdrGetMovieChanges, revised PVS1100 latency information. |

# Licenses

## Microsoft Windows NT™ agreement

Profile Video File Servers contain intellectual property, i.e., software programs that are licensed for use by the end user consumer (hereinafter “End User”).

This is not a sale of intellectual property.

The End User shall not copy, disassemble or reverse-compile the software programs.

THE SOFTWARE PROGRAMS ARE PROVIDED TO THE END USER “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK OF THE QUALITY AND PERFORMANCE OF THE SOFTWARE PROGRAM IS WITH YOU.

THOMSON BROADCAST AND MEDIA SOLUTIONS INC. AND ITS SUPPLIERS SHALL NOT BE HELD TO ANY LIABILITY FOR ANY DAMAGES SUFFERED OF INCURRED BY THE END USER (INCLUDING BUT NOT LIMITED TO, GENERAL, SPECIAL, CONSEQUENTIAL OR INCIDENTAL DAMAGES INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTIONS, LOSS OF BUSINESS INFORMATION AND THE LIKE) ARISING FROM OR IN CONNECTION WITH THE DELIVERY, USE, OR PERFORMANCE OF THE SOFTWARE PROGRAM.

## Independent JPEG software license agreement

The authors make NO WARRANTY or representation, either express or implied, with respect to this software, its quality, accuracy, merchantability, or fitness for a particular purpose. This software is provided “AS IS”, and you, its user, assume the entire risk as to its quality and accuracy.

This software is copyright © 1991, 1992, 1993, 1994, Thomas G. Lane. All Rights Reserved except as specified below.

Permission is hereby granted to use, copy, modify, and distribute this software (or portions thereof) for any purpose, without fee, subject to these conditions: (1) If any part of the source code for this software is distributed, then this README file must be included, with this copyright and no-warranty notice unaltered; and any additions, deletions, or changes to the original files must be clearly indicated in accompanying documentation. (2) If only executable code is distributed, then the accompanying documentation must state that “this software is based in part on the work of the Independent JPEG Group”. (3) Permission for use of this software is granted only if the user accepts full responsibility for any undesirable consequences; the authors accept NO LIABILITY for damages of any kind.

These conditions apply to any software derived from or based on the IJG code, not just to the unmodified library. If you use our work, you ought to acknowledge us.

Permission is NOT granted for the use of any IJG author's name or company name in advertising or publicity relating to this software or products derived from it. This software may be referred to only as “the Independent JPEG Group's software”.

We specifically permit and encourage the use of this software as the basis of commercial products, provided that all warranty or liability claims are assumed by the product vendor.

## Intel GNU general public license agreement

The following listed PDR 100 tools are based on tools from the Intel GNU/960 Tools, some of which were developed and/or distributed by an organization called the Free Software Foundation (FSF): `gdb960.exe` and `objcopy.exe`.

These tools are covered by the GNU General Public License and have no warranty of any kind. The text of this license is built into gdb960.exe, and can be viewed by typing “info copying’ at the `gdb960` prompt. Source code for the above listed tools is available under the terms of this license by contacting Grass Valley.

# Contents


**Chapter 1 Introduction**

Command overview....7
The API libraries....7
Conventions....8
API listings by function....9
TekCfg functionality by category....9
TekPdr functionality by category....10
TekPls functionality by category....14
TekRem functionality by category....17
TekVdr functionality by category....18
TekVfs functionality by category....24
TekVme functionality by category....26
TekXfr functionality by category....28
Profile commands sorted by API....29
Serial command summary....46

**Chapter 2 API Commands**

The Cfg library....64
The Pdr library....85
The Pls library....195
The Rem library....272
The Vdr library....281
The Vfs library....471
The Vme library....514
The Xfr library....559

**Chapter 3 Profile Serial Communication Protocol**

Command structure....564
Reply structure....565
Data representation....566
A typical serial session....567
Configuration files for Profile protocol....569
Overview of configuration files....569
System access commands (0x00)....573
PLS serial protocol commands (0x0E)....595
PLS serial protocol command structure....595
Port access commands (0x10)....647
PDR movie access commands (0x21)....658
Attached movie access commands (0x30)....682
Stored media access commands (0x40)....699
Attached media access commands (0x50)....710
Resource access commands (0x60)....720
Event access commands (0x70)....742
Video mix effects access commands (0x80)....750
Streaming transfer access commands (0x90)....767

**Chapter 4 Odetics, VDCP, and BVW Protocols**

Odetics commands....771
VDCP commands....775
VDCP states....778
BVW commands and responses....782

# List of tables


1 Cfg library commands ..... 29
2 Pdr library commands ..... 30
3 Pls library commands ..... 33
4 Rem library commands ..... 35
5 Vdr library commands ..... 36
6 Vfs library commands ..... 41
7 Vme library commands ..... 43
8 Xfr library commands ..... 45
9 Serial commands: system access ..... 46
10 Serial commands: PLS access ..... 49
11 Serial commands: port access ..... 51
12 Serial commands: PDR movie access ..... 52
13 Serial commands: attached movie access ..... 54
14 Serial commands: stored media access ..... 55
15 Serial commands: attached media access ..... 56
16 Serial commands: resource access ..... 57
17 Serial commands: event access ..... 59
18 Serial commands: video mix effects access ..... 60
19 Serial commands: streaming transfer access ..... 62
20 Vdr resource types ..... 283
21 Vdr events ..... 305
22 Vdr idle states ..... 401
23 Vdr scheduled events ..... 414
25 Supported Odetics commands ..... 772
26 Unsupported Odetics commands ..... 773
27 Supported VDCP commands ..... 775
28 Unsupported VDCP commands ..... 777
29 Port status 1 bits ..... 778
30 VDCP states ..... 779
31 Port status 3 error bits ..... 781
32 Transition from state to state ..... 781
33 Supported BVW commands ..... 782

Chapter **1**

# Introduction

This manual documents the API (Application Programming Interface). It consists of the following chapters:

***Chapter 1,*** *Introduction*. This chapter provides a brief description of the eight API libraries and explains the conventions used in the API command listing. This chapter includes a functional list of each API command as well as tables which sort the commands in two different ways. The tables in *Profile commands sorted by API* on page 29 alphabetically list the API functions along with their equivalent serial commands. The tables in *Serial command summary* on page 46 provide the same information sorted by serial command.

***Chapter 2,*** *API Commands*. This chapter alphabetically lists the routines in the eight API libraries: TekCfg, TekPdr, TekPls, TekRem, TekVdr, TekVfs, TekVme, and TekXfr.

***Chapter 3,*** *Profile Serial Communication Protocol*. This chapter describes the serial specifications and lists these commands by function and number. Each command references the equivalent API command.

***Appendix A,*** *Odetics, VDCP, and BVW Protocols*. This appendix provides information on supported and unsupported third-party protocol commands by Odetics, Louth, and BVW.

For a conceptual overview of the Profile development model, see the *Software Development Kit User Manual.*

## Command overview

### The API libraries

The **TekCfg** library is responsible for providing configuration information such as the number and names of file systems, codecs, media inputs, and media outputs.

The **TekPdr** library is responsible for maintaining the inventory of movies in the common movie format; it also provides an interface by which the movie descriptions can be modified.

The **TekPls** library is responsible for copying files from a Profile to a tape cartridge and back.

The **TekRem** library is responsible for connections between a remote Win32 system and a Profile. It is also responsible for communicating the calls made to the other API libraries residing on the Profile.

The **TekVdr** library is responsible for sharing the embedded system's resources and for the transport control required to record and play multimedia movies.

The **TekVfs** library allows a low level interface to the embedded system's media files.

The **TekVme** library allows control of the video mix effects board.

The **TekXfr** library allows control of streaming, simultaneous movie play/record and movie transfers.

## Conventions

- Values shown in **boldface** should be used exactly as shown.
- Values shown in *italics* indicate a user-supplied value.
- Settings marked as volatile must be stored by the application. They are not saved by the embedded disk recorder system.
- TRUE is defined as a non-zero value; FALSE is defined as 0.
- Types are shown with an initial upper-case letter; variables are shown with an initial lower-case letter.
- Unless otherwise specified, integers are signed (two's-complement) 32-bit numbers.

# API listings by function

## TekCfg functionality by category

### Device configuration

CfgGetCodecAttributes
CfgGetInputAttributes
CfgGetInputName
CfgGetInputUnitNum
CfgGetNumCodecs
CfgGetNumInputs
CfgGetNumOutputs
CfgGetNumTCGenerators
CfgGetNumTCRecorders
CfgGetOutputAttributes
CfgGetOutputName
CfgGetOutputUnitNum
CfgGetStandard
CfgGetVideoInputStandard

### File system information

CfgGetFileSystemName
CfgGetNumFileSystems

### Version information

CfgGetMajorVersion
CfgGetMinorVersion

# TekPdr functionality by category

## Inventory management

### Backup information

PdrBackupMovieDescription
PdrRestoreMovieDescription

### Changing inventory

PdrCopyMovie (PdrGetWaitOpStatus, PdrTerminateWaitOperation)
PdrCopyMovieToken
PdrDeleteMovie
PdrRenameMovie
PdrRenameOpenMovie
PdrSaveMovie (PdrGetWaitOpStatus, PdrTerminateWaitOperation)

### Change notification

PdrCreateEvent
PdrGetMovieChanges

### Enumeration

PdrCloseFind
PdrFindFirstDataset
PdrFindFirstGroup
PdrFindFirstMediaUsage
PdrFindFirstMovie
PdrFindNextDataset
PdrFindNextGroup
PdrFindNextMediaUsage
PdrFindNextMovie
PdrGetListOfNames
PdrGetNumMovies
PdrMovieExists

### Item information

PdrGetMediaStateInfo
PdrGetMovieStateInfo

## Library management

PdrGetMaxTracks
PdrGetRegistry
PdrSetRegistry

## Movie characteristics

PdrGetMovieAttributes
PdrGetMovieCreateTime
PdrGetMovieDataset
PdrGetMovieGroup
PdrGetMovieLastChangeTime
PdrGetMovieLength
PdrGetMovieMarkIn
PdrGetMovieMarkOut
PdrGetMovieName
PdrGetUserData
PdrSetMovieLocked
PdrSetMovieReadOnly
PdrSetMovieReadWrite
PdrSetMovieUnlocked
PdrSetUserData

## Name translation control

PdrGetCurrentDataset
PdrGetCurrentGroup
PdrSetCurrentDataset
PdrSetCurrentGroup

## Stored movie editing and evaluation

### Convenience routines

PdrSetClipIn
PdrSetClipOut
PdrTrimClip

### Location identification

PdrGetFirstTcPosition
PdrGetLastTcPosition
PdrGetPositionOfMediaToken

### Object addition/deletion

PdrCopyMediaToken
PdrCreateBlackMediaToken
PdrCreateMediaToken
PdrCreateMediaTokenWithPath
PdrCreateMovie
PdrCreateTrackToken
PdrDeleteMedia
PdrNullMediaToken

### Object identification

PdrCloseMovie
PdrGetMediaTokenAtPosition
PdrGetNextMediaToken
PdrGetNextTrack
PdrGetNumMediaOnTrack
PdrGetNumTracks
PdrGetPreviousMediaToken
PdrGetPreviousTrack
PdrOpenMovie

### Object information

PdrGetMediaAttributes
PdrGetMediaDataset
PdrGetMediaDataType
PdrGetMediaDuration
PdrGetMediaIn
PdrGetMediaMarks
PdrGetMediaOut
PdrGetMediaPath
PdrGetMediaState
PdrGetMediaTokenType
PdrGetMovieTokenFromMediaToken
PdrGetResourceTypeFromTrackType
PdrGetTrackLength
PdrGetTrackTokenFromMediaToken
PdrGetTrackTokenNum
PdrGetTrackTokenType
PdrGetTrackTypeFromResourceType
PdrMediaTokenIsNull
PdrSetMediaMarkIn
PdrSetMediaMarkOut
PdrSetMediaMarks
PdrSetMovieMarkIn
PdrSetMovieMarkOut
PdrSetMovieMarks

## User data extension management

PdrDeleteExtensionAtPos
PdrFindFirstExtensionPos
PdrFindNextExtensionPos
PdrFreeExtension
PdrGetExtensionAtPos
PdrGetExtensionInfoAtPos
PdrInsertExtension
PdrReadExtension

## Version information

PdrGetMajorVersion
PdrGetMinorVersion

# TekPls functionality by category

## Basic archive operations

PlsArchiveClip
PlsArchiveDataFile
PlsBackupCatalog
PlsCloseBinInfo
PlsCloseFile
PlsCloseFileInfo
PlsCloseFindHandle
PlsConnectFile
PlsDeleteFile
PlsGetBackupDir
PlsGetClipSize
PlsGetFileDescription
PlsRenameFile
PlsRestore
PlsRestoreDataFile
PlsSetBackupDir
PlsSetFileDescription

## Cartridge management

PlsAllocateCartridge
PlsCartStateUpdateNeeded
PlsCloseCartridge
PlsCloseCartridgeInfo
PlsConnectCartridge
PlsCopyCartridge
PlsExportCartridge
PlsFormatCartridge
PlsGetCartDescription
PlsGetCartridgeConfig
PlsGetCartridgeStatus
PlsGetLocationString
PlsGetPartitionMap
PlsImportCartridge
PlsImportLoadCartridge
PlsInventoryCartridge
PlsSetCartDescription
PlsSetLocationString
PlsUpdateCartridge

## Enumeration

PlsFindFirstBinInfo
PlsFindFirstCartridgeInfo
PlsFindFirstFileInfo
PlsFindFirstHandle
PlsFindFirstTransportConfig
PlsFindNextBinInfo
PlsFindNextCartridgeInfo
PlsFindNextFileInfo
PlsFindNextHandle
PlsFindNextTransportConfig

## Event management

PlsGetAnyEvent
PlsGetAsynchEvent
PlsGetEventMask
PlsSetEventMask

## Library management

PlsGetLibraryConfig
PlsGetLibraryStatus
PlsInventoryLibrary
PlsCloseLibrary
PlsOpenLibrary
PlsSetModes
PlsGetModes
PlsGetTimeDate
PlsHouseKeeping
PlsSetEventMask
PlsGetEventMask
PlsGetAnyEvent
PlsGetAsynchEvent
PlsGetCommandEvent
PlsSetPath
PlsGetPath

## Miscellaneous commands

PlsCancelCommand
PlsGetStatusCommand

## Transport management

PlsAddTransport
PlsAllocateTransport
PlsCleanTransport
PlsCloseTransport
PlsCloseTransportConfig
PlsConnectTransport
PlsGetTransportStatus
PlsLoadTransport
PlsRemoveTransport
PlsUnloadTransport

## Version information

PlsGetMajorVersion
PlsGetMinorVersion

# TekRem functionality by category

## Connection management

RemCloseConnection
RemGetConnectionName
RemIsHandleRemote
RemIsolateConnection
RemIsolateHandle
RemOpenConnection

## Version information

RemGetMajorVersion
RemGetMinorVersion

# TekVdr functionality by category

## Event management

VdrCancelEvent
VdrDefaultEvent
VdrGetNextEvent
VdrGetPreviousEvent
VdrRescheduleEvent
VdrScheduleEvent
VdrShiftEvents
VdrWhenIsEvent

## GPI control

VdrCancelAllPendingGpiTriggerCommands
VdrDefaultEventOnGpiInputTrigger
VdrSetGpiOutput
VdrSetGpiOutputOnGpiInputTrigger
VdrShuttleOnGpiInputTrigger
VdrStateEventOnGpiInputTrigger
VdrTriggerGpiOutput
VdrTriggerGpiOutputOnGpiInputTrigger
VdrTriggerGpiOutputAtGenTc
VdrWaitForGpiInputTrigger

## Media control

### Fitted Media

VdrAttachFittedMedia
VdrAttachFittedMediaWithMarks
VdrSetFittedMediaMarkIn
VdrSetFittedMediaMarkOut
VdrSetMediaDuration

### Media attributes

VdrGetMediaDuration
VdrGetMediaEndPosition
VdrGetMediaMarkDuration
VdrGetMediaMarkIn
VdrGetMediaMarkOut
VdrGetMediaName
VdrGetMediaStartPosition
VdrGetMediaUserData
VdrSetMediaDuration
VdrSetMediaMarkIn
VdrSetMediaMarkOut
VdrSetMediaUserData

### Media management

VdrAttachMedia
VdrAttachMediaWithMarks
VdrDetachMedia
VdrGetMediaAtPosition
VdrGetNextMedia
VdrGetPreviousMedia

## Movie control

### Movie attributes

VdrGetMovieAccessMode
VdrGetMovieAtPosition
VdrGetMovieDefaultMarkIn
VdrGetMovieDefaultMarkOut
VdrGetMovieEndPosition
VdrGetMovieMarkDuration
VdrGetMovieMarkIn
VdrGetMovieMarkOut
VdrGetMovieName
VdrGetMovieStartPosition
VdrGetMovieUserData
VdrSetMovieMarkIn
VdrSetMovieMarkOut
VdrSetMovieUserData

### Movie enumeration

VdrGetNextMovie
VdrGetPreviousMovie

### Movie management

VdrAttachMovie
VdrAttachMovieWithMarks
VdrAttachOpenMovie
VdrAttachOpenMovieWithMarks
VdrDetachMovie
VdrExpandMovie

### Position limiting functions

VdrDisableMoviePositionLimiting
VdrEnableMoviePositionLimiting
VdrIsMoviePositionLimitingEnabled

## Port control

### Port attributes

VdrGetCurrentRate
VdrGetCurrentState
VdrGetPlayMode
VdrGetStillPlayMode
VdrGetVideoFormat
VdrSetFieldDominance
VdrSetPlayMode
VdrSetStillPlayMode

### Port enumeration

VdrGetNextPort
VdrGetPortFromEvent
VdrGetPortFromMedia
VdrGetPortFromMovie
VdrGetPortFromResource
VdrGetPreviousPort

### Port management

VdrClosePort
VdrOpenPortConnection
VdrResetPort

### Port position

VdrGetMaxPosition
VdrGetMinPosition
VdrGetPosition
VdrSetMaxPosition
VdrSetMinPosition
VdrSetPosition
VdrSetPositionLimits
VdrSetPositionRelative

### Port state

VdrCuePlay
VdrCueRecord
VdrIdle
VdrJog
VdrShuttle

## Resource control

### Audio-specific resource management

VdrGetAudioInputLevel
VdrGetAudioLevel
VdrGetAudioOutputLevel
VdrGetAudioWindow
VdrSetAudioInputLevel
VdrSetAudioOutputLevel
VdrSetAudioWindow

### Channel management (Central resource management)

VdrGetChannelInfoList
VdrGetNumChannelDefs
VdrAllocateChannel

### Resource management

VdrActivateResource
VdrAllocateResource
VdrDeactivateResource
VdrGetNextResource
VdrGetNextResourceOfType
VdrGetPreviousResource
VdrGetPreviousResourceOfType
VdrGetResourceConnectionHandle
VdrGetResourceFromMedia
VdrGetResourceNumber
VdrGetResourceType
VdrGetResourceUserData
VdrGetVideoCombinerPartHandle
VdrIsResourceActive
VdrIsResourceAvailable
VdrReleaseResource
VdrResetResource
VdrSetResourceUserData

### Timecode-specific resource management

VdrGetCurrentTimeCode
VdrGetCurrentUserBits
VdrGetGenTcFormat
VdrGetGenTcMode
VdrSetGenTcFormat
VdrSetGenTcMode

### Video-specific resource management

VdrGetAbsMaxChrQ
VdrGetAbsMaxLumQ
VdrGetAbsMinChrQ
VdrGetAbsMinLumQ
VdrGetCurrentFieldSize
VdrGetCurrentLumQFactor
VdrGetFieldDominance
VdrGetMaxChrQ
VdrGetMaxLumQ
VdrGetMinChrQ
VdrGetMinLumQ
VdrSetChrQLimits
VdrSetLumQLimits
VdrSetMaxChrQ
VdrSetMaxLumQ
VdrSetMinChrQ
VdrSetMinLumQ

## System information

VdrGetRecordCapacity
VdrGetStateChangeLatency
VdrGetTotalRecordCapacity
VdrGetVideoFieldCount
VdrGetVideoGoalSize
VdrSetVideoFormat
VdrSetVideoGoalSize
VdrWaitForVideoField

## Version information

VdrGetMajorVersion
VdrGetMinorVersion

# TekVfs functionality by category

## Copy/move

VfsCancelCopy
VfsCloseHandle
VfsCopyFile
VfsCopyMediaSegment
VfsMoveFile
VfsMoveMediaSegment
VfsStatusOfCopy

## File attributes

VfsGetFileAttributes
VfsGetFileDefaultMarkIn
VfsGetFileDefaultMarkOut
VfsGetFileDefaultMarks
VfsGetFileModificationTime
VfsGetFileRecordedDuration
VfsGetFileType
VfsGetFileVideoFormat
VfsSetCurrentDirectory
VfsSetFileAttributes

## File/directory management

VfsCreateDirectory
VfsCreateFile
VfsCreateMediaFile
VfsDeleteFile
VfsDirectoryExists
VfsEraseMediaSegment
VfsFileExists
VfsGetCurrentDirectory
VfsGetCurrentDrive

## File enumeration

VfsFindClose
VfsFindFirstFile
VfsFindNextFile

## Read/write

VfsFieldFree
VfsFieldRead
VfsFieldWrite
VfsReadFile
VfsWriteFile

## Version information

VfsGetEngineMajorVersion
VfsGetEngineMinorVersion
VfsGetMajorVersion
VfsGetMinorVersion

# TekVme functionality by category

## Edge and border parameters

VmeGetBorderColor
VmeGetBorderWidth
VmeGetEdgeSoftness
VmeGetHModulation
VmeGetVModulation
VmeSetBorderColor
VmeSetBorderWidth
VmeSetEdgeSoftness
VmeSetHModulation
VmeSetVModulation

## Luminance key parameters

VmeGetKey
VmeSetKey

## Matte parameters

VmeGetFadeMatteColor
VmeSetFadeMatteColor

## Mix contexts

VmeAllocateContext
VmeApplyContext
VmeFreeContext

## Position parameters

VmeGetMultipliers
VmeGetPosition
VmeGetRotation
VmeSetMultipliers
VmeSetPosition
VmeSetRotation

## Timing and initial progress parameters

VmeGetFadeLength
VmeGetTransitionLength
VmeSetFadeLength
VmeSetTransitionLength

## Transitions and fades

VmeGetFadeProgress
VmeGetTransitionProgress
VmeGetTransitionType
VmeResetFade
VmeResetTransition
VmeSetFadeProgress
VmeSetTransitionProgress
VmeSetTransitionType
VmeStartFade
VmeStartTransition

## Version information

VmeGetMajorVersion
VmeGetMinorVersion

# TekXfr functionality by category

## Transfer management

XfrAbort
XfrGetActiveTokens
XfrGetStatus
XfrRequest

# Profile commands sorted by API

**Table 1. Cfg library commands**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *CfgGetCodecAttributes* | 64 | *Get Attributes by Type* | 00.18 | 579 |
| *CfgGetCodecTimingMode* | 65 | *Get Default Codec Timing* | 00.38 | 592 |
| *CfgGetFileSystemName* | 66 | *Get File System Name* | 00.05 | 575 |
| *CfgGetInputAttributes* | 67 | *Get Attributes by Type* | 00.18 | 579 |
| *CfgGetInputName* | 68 | *Get Input Name* | 00.07 | 575 |
| *CfgGetInputUnitNum* | 69 | *Get Input Unit* | 00.09 | 576 |
| *CfgGetMajorVersion* | 70 | *Get TekCfg Major Version Number* | 00.16 | 578 |
| *CfgGetMinorVersion* | 71 | *Get TekCfg Minor Version Number* | 00.17 | 579 |
| *CfgGetNumCodecs* | 72 | *Get Number of Types* | 00.06 | 575 |
| *CfgGetNumFileSystems* | 73 | *Get Number of File Systems* | 00.04 | 574 |
| *CfgGetNumInputs* | 74 | *Get Number of Types* | 00.06 | 575 |
| *CfgGetNumOutputs* | 75 | *Get Number of Types* | 00.06 | 575 |
| *CfgGetNumResources* | 76 | *Get Number of Resources* | 00.3E | 593 |
| *CfgGetNumTCGenerators* | 77 | *Get Number of Types* | 00.06 | 575 |
| *CfgGetNumTCRecorders* | 78 | *Get Number of Types* | 00.06 | 575 |
| *CfgGetOutputAttributes* | 79 | *Get Attributes by Type* | 00.18 | 579 |
| *CfgGetOutputName* | 80 | *Get Output Name* | 00.08 | 576 |
| *CfgGetOutputUnitNum* | 81 | *Get Output Unit* | 00.0A | 576 |
| *CfgGetResourceName* | 82 | *Get Resource Name* | 00.3D | 593 |
| *CfgGetStandard* | 83 | *Get Standard* | 00.0B | 576 |
| *CfgGetVideoInputStandard* | 84 | *Get Video Standard* | 00.0C | 576 |

**Table 2. Pdr library commands**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *PdrBackupMovieDescription* | 85 | *Pdr Backup Movie Descriptor* | 21.20 | 672 |
| *PdrCloseAllEnumTokens* | 86 | *Pdr Close All Enumerators* | 21.31 | 680 |
| *PdrCloseFind* | 87 | *Pdr Close Find* | 21.03 | 660 |
| *PdrCloseMovie* | 88 | *Pdr Close Movie* | 21.0E | 664 |
| *PdrCopyMediaToken* | 89 | *Copy Media Token* | 40.0C | 703 |
| *PdrCopyMovie* | 90 | *Pdr Copy Movie* | 21.18 | 668 |
| *PdrCopyMovieToken* | 91 | *Pdr Copy Movie Token* | 21.0F | 664 |
| *PdrCreateBlackMediaToken* | 92 | *Create Black Media Token* | 40.09 | 702 |
| *PdrCreateEvent* | 93 | *Pdr Create Event* | 21.24 | 674 |
| *PdrCreateMediaToken* | 94 | *Create Media Token* | 40.08 | 702 |
| *PdrCreateMediaTokenWithPath* | 95 | *Create Media Token with Path* | 40.0A | 703 |
| *PdrCreateMovie* | 96 | *Pdr Create Movie* | 21.0C | 663 |
| *PdrCreateTrackToken* | 97 | *Create Track Token* | 40.13 | 705 |
| *PdrDeleteExtensionAtPos* | 98 | *No serial equivalent.* | | |
| *PdrDeleteMedia* | 99 | *Delete Media* | 40.0B | 703 |
| *PdrDeleteMovie* | 100 | *Pdr Delete Movie* | 21.11 | 665 |
| *PdrFindFirstDataset* | 101 | *Pdr Find First Dataset* | 21.01 | 660 |
| *PdrFindFirstExtensionPos* | 102 | *No serial equivalent.* | | |
| *PdrFindFirstGroup* | 103 | *Pdr Find First Group* | 21.06 | 661 |
| *PdrFindFirstMediaUsage* | 104 | *No serial equivalent.* | | |
| *PdrFindFirstMovie* | 105 | *Pdr Find First Movie* | 21.0A | 663 |
| *PdrFindNextDataset* | 106 | *Pdr Find Next Dataset* | 21.02 | 660 |
| *PdrFindNextExtensionPos* | 107 | *No serial equivalent.* | | |
| *PdrFindNextGroup* | 108 | *Pdr Find Next Group* | 21.07 | 662 |
| *PdrFindNextMediaUsage* | 109 | *No serial equivalent.* | | |
| *PdrFindNextMovie* | 110 | *Pdr Find Next Movie* | 21.0B | 663 |
| *PdrFreeExtension* | 111 | *No serial equivalent.* | | |
| *PdrGetCurrentDataset* | 112 | *Pdr Get Current Dataset* | 21.04 | 661 |
| *PdrGetCurrentGroup* | 113 | *Pdr Get Current Group* | 21.08 | 662 |
| *PdrGetExtensionAtPos* | 114 | *No serial equivalent.* | | |
| *PdrGetExtensionInfoAtPos* | 115 | *No serial equivalent.* | | |
| *PdrGetFirstTcPosition* | 116 | *Get First Timecode Position* | 21.27 | 676 |
| *PdrGetFullName* | 117 | *No serial equivalent.* | | |
| *PdrGetLastTcPosition* | 118 | *Get Last Timecode Position* | 21.28 | 676 |
| *PdrGetListOfNames* | 119 | *Pdr Get List of Names* | 21.22 | 673 |
| *PdrGetMajorVersion* | 121 | *Get TekPdr Major Version Number* | 00.1B | 580 |
| *PdrGetMaxTracks* | 122 | *Get Maximum Number of Tracks* | 40.01 | 700 |
| *PdrGetMediaAttributes* | 123 | *Get Stored Media Status* | 40.00 | 699 |
| *PdrGetMediaDataset* | 124 | *Get Stored Media Status* | 40.00 | 699 |

**Table 2. Pdr library commands (Continued)**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *PdrGetMediaDataType* | 125 | *Get Stored Media Status* | 40.00 | 699 |
| *PdrGetMediaDuration* | 126 | *Get Stored Media Status* | 40.00 | 699 |
| *PdrGetMediaIn* | 127 | *Get Stored Media Status* | 40.00 | 699 |
| *PdrGetMediaMarks* | 128 | *Get Stored Media Status* | 40.00 | 699 |
| *PdrGetMediaOut* | 129 | *Get Stored Media Status* | 40.00 | 699 |
| *PdrGetMediaPath* | 130 | *Get Stored Media Status* | 40.00 | 699 |
| *PdrGetMediaStateInfo* | 131 | *Get Media State* | 40.1A | 707 |
| *PdrGetMediaTokenAtPosition* | 133 | *Get Media Token at Position* | 40.0F | 704 |
| *PdrGetMediaTokenType* | 134 | *Get Stored Media Status* | 40.00 | 699 |
| *PdrGetMinorVersion* | 135 | *Get TekPdr Minor Version Number* | 00.1C | 580 |
| *PdrGetMovieAttributes* | 136 | *Pdr Get Movie Status* | 21.00 | 659 |
| *PdrGetMovieChanges* | 137 | *Pdr Get Movie Changes* | 21.23 | 674 |
| *PdrGetMovieCreateTime* | 139 | *Pdr Get Movie Status* | 21.00 | 659 |
| *PdrGetMovieDataset* | 140 | *Pdr Get Movie Status* | 21.00 | 659 |
| *PdrGetMovieGroup* | 141 | *Pdr Get Movie Status* | 21.00 | 659 |
| *PdrGetMovieLastChangeTime* | 142 | *Pdr Get Movie Status* | 21.00 | 659 |
| *PdrGetMovieLength* | 143 | *Pdr Get Movie Status* | 21.00 | 659 |
| *PdrGetMovieMarkIn* | 144 | *No serial equivalent.* | | |
| *PdrGetMovieMarkOut* | 145 | *No serial equivalent.* | | |
| *PdrGetMovieName* | 146 | *Pdr Get Movie Status* | 21.00 | 659 |
| *PdrGetMovieStateInfo* | 147 | *Pdr Get Movie State* | 21.12 | 666 |
| *PdrGetMovieTokenFromMediaToken* | 148 | *Get Stored Media Status* | 40.00 | 699 |
| *PdrGetNextMediaToken* | 149 | *Get Next Media Token* | 40.0D | 703 |
| *PdrGetNextTrack* | 150 | *Get Next Track* | 40.03 | 701 |
| *PdrGetNumMediaOnTrack* | 151 | *Get Number of Media on Track* | 40.07 | 702 |
| *PdrGetNumMovies* | 152 | *Pdr Get Number of Movies* | 21.14 | 667 |
| *PdrGetNumTracks* | 153 | *Get Number of Tracks* | 40.02 | 701 |
| *PdrGetPositionOfMediaToken* | 154 | *Get Stored Media Status* | 40.00 | 699 |
| *PdrGetPreviousMediaToken* | 155 | *Get Previous Media Token* | 40.0E | 704 |
| *PdrGetPreviousTrack* | 156 | *Get Previous Track* | 40.04 | 701 |
| *PdrGetRegistry* | 157 | *Pdr Get Registry* | 21.26 | 675 |
| *PdrGetResourceTypeFromTrackType* | 158 | *Get Resource Type From Track Type* | 40.17 | 706 |
| *PdrGetTrackLength* | 159 | *Get Track Length* | 40.05 | 701 |
| *PdrGetTrackTokenFromMediaToken* | 160 | *Get Track Token From Media Token* | 40.06 | 702 |
| *PdrGetTrackTokenNum* | 161 | *Get Track Token Number* | 40.14 | 705 |
| *PdrGetTrackTokenType* | 162 | *Get Track Token Type* | 40.15 | 706 |
| *PdrGetTrackTypeFromResourceType* | 163 | *Get Track Type From Resource* | 40.16 | 706 |
| *PdrGetUserData* | 164 | *Pdr Get User Data* | 21.1E | 671 |
| | | *Pdr Get User Data Commit* | 21.1F | 672 |

**Table 2. Pdr library commands (Continued)**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *PdrGetWaitOpStatus* | 165 | *Pdr Get Wait Operation Status* | 21.1A | 669 |
| *PdrInsertExtension* | 166 | *No serial equivalent.* | | |
| *PdrMediaTokenIsNull* | 167 | *Is Media Token Null* | 40.19 | 706 |
| *PdrMovieExists* | 168 | *Pdr Movie Exists* | 21.15 | 667 |
| *PdrNullMediaToken* | 169 | *Get Null Media Token* | 40.18 | 706 |
| *PdrOpenMovie* | 170 | *Pdr Open Movie* | 21.0D | 664 |
| *PdrReadExtension* | 171 | *No serial equivalent.* | | |
| *PdrRenameMovie* | 172 | *Pdr Rename Movie* | 21.10 | 665 |
| *PdrRenameOpenMovie* | 173 | *Rename Open Movie* | 21.29 | 676 |
| *PdrRestoreMovieDescription* | 174 | *Pdr Restore Movie Descriptor* | 21.21 | 673 |
| *PdrSaveMovie* | 175 | *Pdr Save Movie* | 21.19 | 669 |
| | | *Pdr Save Movie By Port* | 21.30 | 679 |
| *PdrSetClipIn* | 176 | *Pdr Set Clip Marks* | 21.16 | 667 |
| *PdrSetClipOut* | 177 | *Pdr Set Clip Marks* | 21.16 | 667 |
| *PdrSetCurrentDataset* | 178 | *Pdr Set Current Dataset* | 21.05 | 661 |
| *PdrSetCurrentGroup* | 179 | *Pdr Set Current Group* | 21.09 | 662 |
| *PdrSetMediaIn* | 180 | *Set Media Mark In* | 40.10 | 704 |
| *PdrSetMediaMarks* | 181 | *Set Media Marks* | 40.12 | 705 |
| *PdrSetMediaOut* | 182 | *Set Media Mark Out* | 40.11 | 704 |
| *PdrSetMovieLocked* | 183 | *Pdr Set Movie Attributes* | 21.13 | 666 |
| | | *Pdr Get Number of Movies* | 21.14 | 667 |
| *PdrSetMovieMarkIn* | 184 | *No serial equivalent.* | | |
| *PdrSetMovieMarkOut* | 185 | *No serial equivalent.* | | |
| *PdrSetMovieMarks* | 186 | *No serial equivalent.* | | |
| *PdrSetMovieReadOnly* | 187 | *Pdr Set Movie Attributes* | 21.13 | 666 |
| *PdrSetMovieReadWrite* | 188 | *Pdr Set Movie Attributes* | 21.13 | 666 |
| *PdrSetMovieUnlocked* | 189 | *Pdr Set Movie Attributes* | 21.13 | 666 |
| *PdrSetRegistry* | 190 | *Pdr Set Registry* | 21.25 | 675 |
| *PdrSetUserData* | 192 | *Pdr Set User Data* | 21.1C | 670 |
| *PdrTerminateWaitOperation* | 193 | *Pdr Terminate Wait Operation* | 21.1B | 670 |
| *PdrTrimClip* | 194 | *Pdr Trim Clip* | 21.17 | 668 |

**Table 3. Pls library commands**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *PlsAddTransport* | 195 | *PlsAddTransport* | 0E.7A | 643 |
| *PlsAllocateCartridge* | 196 | *PlsAllocateCartridge* | 0E.32 | 618 |
| *PlsAllocateTransport* | 197 | *PlsAllocateTransport* | 0E.21 | 614 |
| *PlsArchiveClip* | 198 | *PlsArchiveClip* | 0E.53 | 625 |
| *PlsArchiveDataFile* | 199 | *PlsArchiveDataFile* | 0E.54 | 626 |
| *PlsBackupCatalog* | 200 | *PlsBackupCatalogCommand* | 0E.62 | 629 |
| *PlsCancelCommand* | 201 | *PlsCancelCommand* | 0E.7E | 645 |
| *PlsCleanTransport* | 202 | *PlsCleanTransport* | 0E.26 | 617 |
| *PlsCloseBinInfo* | 203 | *PlsCloseBinInfo* | 0E.0C | 605 |
| *PlsCloseCartridge* | 204 | *PlsCloseCartridge* | 0E.33 | 618 |
| *PlsCloseCartridgeInfo* | 205 | *PlsCloseCartridgeInfo* | 0E.0F | 609 |
| *PlsCloseFile* | 206 | *PlsCloseFile* | 0E.51 | 625 |
| *PlsCloseFileInfo* | 207 | *PlsCloseFileInfo* | 0E.12 | 611 |
| *PlsCloseFindHandle* | 208 | *PlsCloseHandle* | 0E.15 | 613 |
| *PlsCloseLibrary* | 209 | *PlsCloseLibrary* | 0E.04 | 598 |
| *PlsCloseTransport* | 210 | *PlsCloseTransport* | 0E.22 | 614 |
| *PlsCloseTransportConfig* | 211 | *PlsCloseTransportConfig* | 0E.09 | 603 |
| *PlsConnectCartridge* | 212 | *PlsConnectCartridge* | 0E.30 | 618 |
| *PlsConnectFile* | 213 | *PlsConnectFile* | 0E.50 | 625 |
| *PlsConnectTransport* | 214 | *PlsConnectTransport* | 0E.20 | 614 |
| *PlsCopyCartridge* | 215 | *PlsCopyCartridge* | 0E.3A | 622 |
| *PlsDeleteFile* | 216 | *PlsDeleteFile* | 0E.58 | 627 |
| *PlsExportCartridge* | 217 | *PlsExportCartridge* | 0E.3C | 622 |
| *PlsFindFirstBinInfo* | 218 | *PlsFindFirstBinInfo* | 0E.0A | 604 |
| *PlsFindFirstCartridgeInfo* | 219 | *PlsFindFirstCartridgeInfo* | 0E.0D | 606 |
| *PlsFindFirstFileInfo* | 221 | *PlsFindFirstFileInfo* | 0E.10 | 609 |
| *PlsFindFirstHandle* | 223 | *PlsFindFirstHandle* | 0E.13 | 611 |
| *PlsFindFirstTransportConfig* | 224 | *PlsFindFirstTransportConfig* | 0E.07 | 601 |
| *PlsFindNextBinInfo* | 225 | *PlsFindNextBinInfo* | 0E.0B | 605 |
| *PlsFindNextCartridgeInfo* | 226 | *PlsFindNextCartridgeInfo* | 0E.0E | 608 |
| *PlsFindNextFileInfo* | 227 | *PlsFindNextFileInfo* | 0E.11 | 610 |
| *PlsFindNextHandle* | 228 | *PlsFindNextHandle* | 0E.14 | 613 |
| *PlsFindNextTransportConfig* | 229 | *PlsFindNextTransportConfig* | 0E.08 | 603 |
| *PlsFormatCartridge* | 230 | *PlsFormatCartridge* | 0E.38 | 621 |
| *PlsGetAnyEvent* | 231 | *PlsGetAnyEvent* | 0E.72 | 630 |
| *PlsGetAsynchEvent* | 232 | *PlsGetAsynchEvent* | 0E.73 | 634 |
| *PlsGetBackupDir* | 233 | *PlsGetBackupDirCommand* | 0E.61 | 628 |
| *PlsGetCartDescription* | 234 | *PlsGetCartDescription* | 0E.41 | 624 |
| *PlsGetCartridgeConfig* | 235 | *PlsGetCartridgeConfig* | 0E.34 | 619 |

**Table 3. Pls library commands (Continued)**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *PlsGetCartridgeStatus* | 236 | *PlsGetCartridgeStatus* | 0E.35 | 620 |
| *PlsGetClipSize* | 237 | *PlsGetClipSize* | 0E.63 | 629 |
| *PlsGetCommandEvent* | 238 | *PlsGetCommandEvent* | 0E.74 | 638 |
| *PlsGetEventMask* | 239 | *PlsGetEventMask* | 0E.71 | 630 |
| *PlsGetFileDescription* | 240 | *PlsGetFileDescription* | 0E.5B | 627 |
| *PlsGetLibraryConfig* | 241 | *PlsGetLibraryConfig* | 0E.05 | 599 |
| *PlsGetLibraryStatus* | 242 | *PlsGetLibraryStatus* | 0E.06 | 600 |
| *PlsGetLocationString* | 243 | *PlsGetLocationString* | 0E.3E | 623 |
| *PlsGetMajorVersion* | 244 | *PlsGetMajorVersion* | 0E.01 | 597 |
| *PlsGetMinorVersion* | 245 | *PlsGetMinorVersion* | 0E.02 | 597 |
| *PlsGetModes* | 246 | *PlsGetModes* | 0E.76 | 642 |
| *PlsGetPartitionMap* | 247 | *PlsGetPartitionMap* | 0E.43 | 624 |
| *PlsGetPath* | 248 | *PlsGetPath* | 0E.78 | 643 |
| *PlsGetStatusCommand* | 249 | *PlsGetStatusCommand* | 0E.7D | 644 |
| *PlsGetTimeDate* | 250 | *PlsGetTimeDate* | 0E.79 | 643 |
| *PlsGetTransportStatus* | 251 | *PlsGetTransportStatus* | 0E.23 | 615 |
| *PlsHouseKeeping* | 252 | *PlsHouseKeeping* | 0E.7F | 645 |
| *PlsImportCartridge* | 253 | *PlsImportCartridge* | 0E.3D | 622 |
| *PlsImportLoadCartridge* | 254 | *PlsImportLoadCartridge* | 0E.42 | 624 |
| *PlsInventoryCartridge* | 255 | *PlsInventoryCartridge* | 0E.36 | 621 |
| *PlsInventoryLibrary* | 256 | *PlsInventoryLibrary* | 0E.7C | 644 |
| *PlsLoadTransport* | 257 | *PlsLoadTransport* | 0E.24 | 617 |
| *PlsOpenLibrary* | 258 | *PlsOpenLibrary* | 0E.03 | 598 |
| *PlsRemoveTransport* | 259 | *PlsRemoveTransport* | 0E.7B | 644 |
| *PlsRenameFile* | 260 | *PlsRenameFile* | 0E.57 | 626 |
| *PlsRestore* | 261 | *PlsRestore* | 0E.56 | 626 |
| *PlsRestoreDataFile* | 262 | *PlsRestoreDataFile* | 0E.5F | 628 |
| *PlsSetBackupDir* | 263 | *PlsSetBackupDirCommand* | 0E.60 | 628 |
| *PlsSetCartDescription* | 264 | *PlsSetCartDescription* | 0E.40 | 623 |
| *PlsSetEventMask* | 265 | *PlsSetEventMask* | 0E.70 | 629 |
| *PlsSetFileDescription* | 266 | *PlsSetFileDescription* | 0E.5A | 627 |
| *PlsSetLocationString* | 267 | *PlsSetLocationString* | 0E.3F | 623 |
| *PlsSetModes* | 268 | *PlsSetModes* | 0E.75 | 642 |
| *PlsSetPath* | 269 | *PlsSetPath* | 0E.77 | 642 |
| *PlsUnloadTransport* | 270 | *PlsUnloadTransport* | 0E.25 | 617 |
| *PlsUpdateCartridge* | 271 | *PlsUpdateCartridge* | 0E.37 | 621 |

**Table 4. Rem library commands**

| API command name | Page | Serial equivalent |
|---|---|---|
| *RemCloseConnection* | 272 | *No serial equivalent.* |
| *RemGetConnectionName* | 273 | *No serial equivalent.* |
| *RemGetMajorVersion* | 274 | *No serial equivalent.* |
| *RemGetMinorVersion* | 275 | *No serial equivalent.* |
| *RemGetNodeName* | 276 | *No serial equivalent.* |
| *RemIsHandleRemote* | 277 | *No serial equivalent.* |
| *RemIsolateConnection* | 278 | *No serial equivalent.* |
| *RemIsolateHandle* | 279 | *No serial equivalent.* |
| *RemOpenConnection* | 280 | *No serial equivalent.* |

**Table 5. Vdr library commands**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *VdrActivateResource* | 281 | *Activate Resource* | 60.08 | 731 |
| *VdrAllocateChannel* | 282 | *No serial equivalent.* | | |
| *VdrAllocateResource* | 283 | *Allocate Resource* | 60.02 | 726 |
| *VdrAttachFittedMedia* | 287 | *Attach Media* | 50.01 | 711 |
| *VdrAttachFittedMediaWithMarks* | 288 | *Attach Media With Marks* | 50.02 | 712 |
| *VdrAttachMedia* | 289 | *Attach Media* | 50.01 | 711 |
| *VdrAttachMediaWithMarks* | 290 | *Attach Media With Marks* | 50.02 | 712 |
| *VdrAttachMovie* | 291 | *Attach Movie* | 30.01 | 684 |
| *VdrAttachMovieWithMarks* | 293 | *Attach Movie with Marks* | 30.02 | 685 |
| *VdrAttachOpenMovie* | 295 | *Attach Open Movie* | 30.0E | 692 |
| | | *Attach Open Movie By Port* | 30.15 | 697 |
| *VdrAttachOpenMovieWithMarks* | 296 | *Attach Open Movie With Marks* | 30.13 | 695 |
| | | *Attach Open Movie With Marks By Port* | 30.16 | 698 |
| *VdrCancelAllPendingGpiTrigger-Commands* | 297 | *No serial equivalent.* | | |
| *VdrCancelEvent* | 298 | *Cancel Event* | 70.04 | 745 |
| | | *Clear Timeline* | 60.14 | 737 |
| *VdrCancelPendingTimecodeTrigger-Commands* | 299 | *Cancel At Gen Tc Cmds* | 10.15 | 657 |
| *VdrClosePort* | 300 | *Close Port* | 10.02 | 649 |
| *VdrCuePlay* | 301 | *Cue Play* | 10.04 | 650 |
| *VdrCueRecord* | 302 | *Cue Record* | 10.07 | 652 |
| *VdrDeactivateResource* | 303 | *Deactivate Resource* | 60.09 | 732 |
| *VdrDefaultEvent* | 304 | *Default Event* | 70.00 | 743 |
| *VdrDefaultEventAtGenTc* | 307 | *Default Event At Gen Tc* | 70.0B | 747 |
| *VdrDefaultEventOnGpiInputTrigger* | 308 | *No serial equivalent.* | | |
| *VdrDetachMedia* | 309 | *Detach Media* | 50.03 | 713 |
| | | *Clear Timeline* | 60.14 | 737 |
| *VdrDetachMovie* | 310 | *Detach Movie* | 30.03 | 686 |
| | | *Clear Timeline* | 60.14 | 737 |
| *VdrDisableMoviePositionLimiting* | 311 | *Disable Movie Position Limits* | 30.0F | 692 |
| *VdrEnableMoviePositionLimiting* | 312 | *Enable Movie Position Limits* | 30.10 | 693 |
| *VdrExpandMovie* | 313 | *Expand Movie* | 30.0C | 691 |
| | | *Expand Movie By Port* | 30.14 | 696 |
| *VdrGetAbsMaxChrQ* | 314 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetAbsMaxLumQ* | 315 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetAbsMinChrQ* | 316 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetAbsMinLumQ* | 317 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetAudioInputLevel* | 318 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetAudioLevel* | 319 | *Get Resource Status* | 60.00 | 721 |

**Table 5. Vdr library commands (Continued)**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *VdrGetAudioOutputLevel* | 320 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetAudioWindow* | 321 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetBitRate* | 322 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetChannelInfoList* | 323 | *No serial equivalent.* | | |
| *VdrGetCurrentFieldSize* | 324 | *Get Resource Status* | 60.00 | 721 |
| *VdrGetCurrentLumQFactor* | 325 | *Get Resource Status* | 60.00 | 721 |
| *VdrGetCurrentPictureStatus* | 326 | *No serial equivalent.* | | |
| *VdrGetCurrentRate* | 327 | *Get Port Status* | 10.00 | 647 |
| *VdrGetCurrentState* | 328 | *Get Port Status* | 10.00 | 647 |
| *VdrGetCurrentTimeCode* | 329 | *Get Resource Status* | 60.00 | 721 |
| *VdrGetCurrentUserBits* | 330 | *Get Resource Status* | 60.00 | 721 |
| *VdrGetEncodingRange* | 331 | *No serial equivalent.* | | |
| *VdrGetFieldDominance* | 332 | *Get Port Status* | 10.00 | 647 |
| *VdrGetGenTcFormat* | 333 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetGenTcMode* | 334 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetMajorVersion* | 335 | *Get TekVdr Major Version Number* | 00.10 | 577 |
| *VdrGetMaxChrQ* | 336 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetMaxLumQ* | 337 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetMaxPosition* | 338 | *Get Position Limits* | 10.0E | 656 |
| *VdrGetMediaAtPosition* | 339 | *Get Media at Position* | 50.06 | 714 |
| *VdrGetMediaAudioFades* | 340 | *Get Media Audio Fades* | 50.0D | 717 |
| *VdrGetMediaDuration* | 341 | *Get Attached Media Status* | 50.00 | 710 |
| *VdrGetMediaEndPosition* | 342 | *Get Attached Media Status* | 50.00 | 710 |
| *VdrGetMediaMarkDuration* | 343 | *Get Attached Media Status* | 50.00 | 710 |
| *VdrGetMediaMarkIn* | 344 | *Get Attached Media Status* | 50.00 | 710 |
| *VdrGetMediaMarkOut* | 345 | *Get Attached Media Status* | 50.00 | 710 |
| *VdrGetMediaName* | 346 | *Get Attached Media Status* | 50.00 | 710 |
| *VdrGetMediaStartPosition* | 347 | *Get Attached Media Status* | 50.00 | 710 |
| *VdrGetMediaUserData* | 348 | *Get Attached Media Status* | 50.00 | 710 |
| *VdrGetMinChrQ* | 349 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetMinLumQ* | 350 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetMinorVersion* | 351 | *Get TekVdr Minor Version Number* | 00.11 | 577 |
| *VdrGetMinPosition* | 352 | *Get Position Limits* | 10.0E | 656 |
| *VdrGetMovieAccessMode* | 353 | *Get Attached Movie Status* | 30.00 | 682 |
| *VdrGetMovieAtPosition* | 354 | *Get Movie at Position* | 30.06 | 687 |
| *VdrGetMovieDefaultMarkIn* | 355 | *Get Attached Movie Status* | 30.00 | 682 |
| *VdrGetMovieDefaultMarkOut* | 356 | *Get Attached Movie Status* | 30.00 | 682 |
| *VdrGetMovieEndPosition* | 357 | *Get Attached Movie Status* | 30.00 | 682 |
| *VdrGetMovieMarkDuration* | 358 | *Get Attached Movie Status* | 30.00 | 682 |

**Table 5. Vdr library commands (Continued)**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *VdrGetMovieMarkIn* | 359 | *Get Attached Movie Status* | 30.00 | 682 |
| *VdrGetMovieMarkOut* | 360 | *Get Attached Movie Status* | 30.00 | 682 |
| *VdrGetMovieName* | 361 | *Get Attached Movie Status* | 30.00 | 682 |
| *VdrGetMovieStartPosition* | 362 | *Get Attached Movie Status* | 30.00 | 682 |
| *VdrGetMovieUserData* | 363 | *Get Attached Movie Status* | 30.00 | 682 |
| *VdrGetMpegChromaFormat* | 364 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetMpegGopLimits* | 365 | *Get Mpeg Gop Limits* | 60.1E | 740 |
| *VdrGetMpegGopStructure* | 366 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetNextEvent* | 367 | *Get Next Event Handle* | 70.09 | 746 |
| *VdrGetNextMedia* | 368 | *Get Next Media* | 50.04 | 713 |
| *VdrGetNextMovie* | 369 | *Get Next Movie* | 30.04 | 686 |
| *VdrGetNextPort* | 370 | *Get Next Open Port* | 10.0F | 656 |
| *VdrGetNextResource* | 371 | *Get Next Resource* | 60.06 | 730 |
| *VdrGetNextResourceOfType* | 372 | *Get Next Resource* | 60.06 | 730 |
| *VdrGetNumChannelDefs* | 373 | *No serial equivalent.* | | |
| *VdrGetPlayMode* | 374 | *Get Port Status* | 10.00 | 647 |
| *VdrGetPortFromEvent* | 375 | *Get Port Handle from Event Handle* | 70.08 | 746 |
| *VdrGetPortFromMedia* | 376 | *Get Port Handle From Media Handle* | 50.0A | 716 |
| *VdrGetPortFromMovie* | 377 | *Get Port Handle from Movie Handle* | 30.09 | 689 |
| *VdrGetPortFromResource* | 378 | *Get Port Handle from Resource Handle* | 60.13 | 737 |
| *VdrGetPosition* | 379 | *Get Port Status* | 10.00 | 647 |
| *VdrGetPreviousEvent* | 380 | *Get Previous Event Handle* | 70.0A | 746 |
| *VdrGetPreviousMedia* | 381 | *Get Previous Media* | 50.05 | 713 |
| *VdrGetPreviousMovie* | 382 | *Get Previous Movie* | 30.05 | 687 |
| *VdrGetPreviousPort* | 383 | *Get Previous Open Port* | 10.10 | 656 |
| *VdrGetPreviousResource* | 384 | *Get Previous Resource* | 60.07 | 730 |
| *VdrGetPreviousResourceOfType* | 385 | *Get Previous Resource* | 60.07 | 730 |
| *VdrGetRecordCapacity* | 386 | *Get Disk Capacity* | 00.03 | 574 |
| *VdrGetResourceConnectionHandle* | 387 | *Get Resource Connect Handle* | 60.03 | 728 |
| *VdrGetResourceFromMedia* | 388 | *Get Resource From Media* | 50.0C | 717 |
| *VdrGetResourceNumber* | 389 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetResourceType* | 390 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetResourceUserData* | 391 | *Get Resource User Data* | 60.16 | 738 |
| *VdrGetSavedVbiLines* | 392 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetStateChangeLatency* | 393 | *No serial equivalent.* | | |
| *VdrGetStillPlayMode* | 394 | *Get Port Status* | 10.00 | 647 |
| *VdrGetTotalRecordCapacity* | 395 | *Get Disk Capacity* | 00.03 | 574 |
| *VdrGetVideoCodecTiming* | 396 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrGetVideoCombinerPartHandle* | 397 | *Get Video Combiner Part Handle* | 60.17 | 738 |

**Table 5. Vdr library commands (Continued)**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *VdrGetVideoFieldCount* | 398 | *Get Video Field Count* | 00.19 | 580 |
| *VdrGetVideoFormat* | 399 | *Get Video Format* | 00.14 | 578 |
| *VdrGetVideoGoalSize* | 400 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrIdle* | 401 | *Idle* | 10.03 | 650 |
| *VdrIsMoviePositionLimitingEnabled* | 403 | *Is Movie Position Limiting Enabled* | 30.11 | 693 |
| *VdrIsResourceActive* | 404 | *Get Resource Configuration* | 60.01 | 723 |
| *VdrIsResourceAvailable* | 405 | *Is Resource Available* | 60.12 | 737 |
| *VdrJog* | 406 | *Jog Play* | 10.06 | 651 |
| | | *Jog Record* | 10.09 | 653 |
| *VdrMergeResources* | 407 | *Merge Resources* | 60.1B | 739 |
| *VdrOpenPortConnection* | 408 | *Open Port* | 10.01 | 649 |
| *VdrReleaseResource* | 409 | *Release Resource* | 60.04 | 729 |
| *VdrRescheduleEvent* | 410 | *Reschedule Event* | 70.02 | 745 |
| *VdrResetPort* | 411 | *Reset Port* | 10.12 | 657 |
| *VdrResetResource* | 412 | *Reset Resource* | 60.05 | 729 |
| *VdrScheduleEvent* | 413 | *Schedule Event* | 70.01 | 744 |
| *VdrSetAudioInputLevel* | 416 | *Set Audio Input Level* | 60.0D | 734 |
| *VdrSetAudioOutputLevel* | 417 | *Set Audio Output Level* | 60.0E | 735 |
| *VdrSetAudioWindow* | 418 | *Set Audio Window* | 60.0F | 735 |
| *VdrSetBitRate* | 419 | *Set Bit-Rate* | 60.1D | 739 |
| *VdrSetChrQLimits* | 420 | *Set Chrominance Q Limits* | 60.0C | 734 |
| *VdrSetDefaultTcFormat* | 421 | *No serial equivalent.* | | |
| *VdrSetEncodingRange* | 422 | *No serial equivalent.* | | |
| *VdrSetFieldDominance* | 423 | *Set Field Dominance* | 10.11 | 657 |
| *VdrSetFittedMediaMarkIn* | 424 | *Set Media Mark In* | 50.07 | 714 |
| *VdrSetFittedMediaMarkOut* | 425 | *Set Media Mark Out* | 50.08 | 715 |
| *VdrSetGenTcFormat* | 426 | *Set Timecode Generator Format* | 60.10 | 736 |
| *VdrSetGenTcMode* | 427 | *Set Timecode Generator Mode* | 60.11 | 736 |
| *VdrSetGenTcRegenSource* | 428 | *Set Gen Tc Regen Source* | 60.20 | 740 |
| *VdrSetGpiOutput* | 429 | *No serial equivalent.* | | |
| *VdrSetGpiOutputOnGpiInputTrigger* | 430 | *No serial equivalent.* | | |
| *VdrSetLumQLimits* | 431 | *Set Luminance Q Limits* | 60.0B | 733 |
| *VdrSetMaxChrQ* | 432 | *Set Chrominance Q Limits* | 60.0C | 734 |
| *VdrSetMaxLumQ* | 433 | *Set Luminance Q Limits* | 60.0B | 733 |
| *VdrSetMaxPosition* | 434 | *Set Position Limits* | 10.0D | 655 |
| *VdrSetMediaAudioFades* | 435 | *Set Media Audio Fades* | 50.0E | 717 |
| *VdrSetMediaDuration* | 436 | *Set Media Duration* | 50.09 | 716 |
| *VdrSetMediaMarkIn* | 437 | *Set Media Mark In* | 50.07 | 714 |
| *VdrSetMediaMarkOut* | 438 | *Set Media Mark Out* | 50.08 | 715 |

**Table 5. Vdr library commands (Continued)**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *VdrSetMediaUserData* | 439 | *Set Media User Data* | 50.0B | 716 |
| *VdrSetMinChrQ* | 440 | *Set Chrominance Q Limits* | 60.0C | 734 |
| *VdrSetMinLumQ* | 441 | *Set Luminance Q Limits* | 60.0B | 733 |
| *VdrSetMinPosition* | 442 | *Set Position Limits* | 10.0D | 655 |
| *VdrSetMovieMarkIn* | 443 | *Set Movie Mark In* | 30.07 | 688 |
| *VdrSetMovieMarkOut* | 444 | *Set Movie Mark Out* | 30.08 | 689 |
| *VdrSetMovieUserData* | 445 | *Set Movie User Data* | 30.0A | 690 |
| *VdrSetMpegChromaFormat* | 446 | *Set Mpeg Chroma Format* | 60.19 | 739 |
| *VdrSetMpegGopStructure* | 447 | *Set Mpeg Gop Structure* | 60.1A | 739 |
| *VdrSetPlayMode* | 448 | *Set Motion Play/Record Mode* | 10.0B | 654 |
| *VdrSetPosition* | 450 | *Set Position* | 10.0A | 653 |
| *VdrSetPositionLimits* | 451 | *Set Position Limits* | 10.0D | 655 |
| *VdrSetPositionRelative* | 452 | *Set Position* | 10.0A | 653 |
| *VdrSetResourceUserData* | 453 | *Set Resource User Data* | 60.15 | 738 |
| *VdrSetSavedVbiLines* | 454 | *Set Saved Vbi Lines* | 60.21 | 740 |
| *VdrSetStillPlayMode* | 455 | *Set Still Play Mode* | 10.0C | 655 |
| *VdrSetVideoCodecTiming* | 456 | *Set Video Codec Timing* | 60.18 | 738 |
| *VdrSetVideoFormat* | 457 | *Set Video Format* | 00.15 | 578 |
| *VdrSetVideoGoalSize* | 458 | *Set Video Goal Size* | 60.0A | 733 |
| *VdrShuttle* | 459 | *Shuttle Play* | 10.05 | 651 |
| | | *Shuttle Record* | 10.08 | 652 |
| *VdrShuttleAtGenTc* | 460 | *Shuttle At Gen Tc* | 10.14 | 657 |
| *VdrShuttleOnGpiInputTrigger* | 461 | *No serial equivalent.* | | |
| *VdrStateEvent* | 462 | *State Event* | 70.0C | 747 |
| *VdrStateEventAtGenTc* | 463 | *Default Event At Gen Tc* | 70.0D | 747 |
| *VdrStateEventOnGpiInputTrigger* | 464 | *No serial equivalent.* | | |
| *VdrTriggerGpiOutput* | 465 | *No serial equivalent.* | | |
| *VdrTriggerGpiOutputOnGpiInput-Trigger* | 466 | *No serial equivalent.* | | |
| *VdrTriggerGpiOutputAtGenTc* | 467 | *No serial equivalent.* | | |
| *VdrWaitForGpiInputTrigger* | 468 | *No serial equivalent.* | | |
| *VdrWaitForVideoField* | 469 | *Wait For Video Field* | 00.1A | 580 |
| *VdrWhenIsEvent* | 470 | *When Is Event* | 70.05 | 746 |

**Table 6. Vfs library commands**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *VfsCancelCopy* | 471 | *Vfs Cancel Copy* | 00.32 | 590 |
| *VfsCloseHandle* | 472 | *Vfs Close Handle* | 00.2A | 587 |
| *VfsCopyFile* | 473 | *Vfs Copy File* | 00.2E | 588 |
| *VfsCopyMediaSegment* | 474 | *Vfs Copy Media Segment* | 00.2F | 588 |
| *VfsCreateDirectory* | 475 | *Create Directory* | 00.20 | 581 |
| *VfsCreateFile* | 476 | *Vfs Create File* | 00.23 | 583 |
| *VfsCreateMediaFile* | 477 | *Vfs Create Media File* | 00.22 | 583 |
| *VfsDeleteFile* | 478 | *Vfs Delete File* | 00.24 | 584 |
| *VfsDirectoryExists* | 479 | *Directory Exists* | 00.1F | 581 |
| *VfsEraseMediaSegment* | 480 | *Vfs Erase Media Segment* | 00.29 | 586 |
| *VfsFieldFree* | 481 | *Vfs Field Free* | 00.35 | 591 |
| *VfsFieldRead* | 482 | *Vfs Field Read* | 00.33 | 590 |
| | | *Vfs Field Read Data* | 00.34 | 591 |
| *VfsFieldWrite* | 483 | *Vfs Field Write Data* | 00.36 | 591 |
| | | *Vfs Field Write Commit* | 00.37 | 592 |
| *VfsFileExists* | 484 | *Vfs Get File Status* | 00.21 | 582 |
| *VfsFindClose* | 485 | *Vfs Find Close* | 00.2D | 588 |
| *VfsFindFirstFile* | 486 | *Vfs Find First File* | 00.2B | 587 |
| *VfsFindNextFile* | 487 | *Vfs Find Next File* | 00.2C | 587 |
| *VfsGetCurrentDirectory* | 488 | *Get Current Directory* | 00.0D | 577 |
| *VfsGetCurrentDrive* | 489 | *Get Current Drive* | 00.0F | 577 |
| *VfsGetEngineMajorVersion* | 490 | *Get Engine Major Version Number* | 00.12 | 578 |
| *VfsGetEngineMinorVersion* | 491 | *Get Engine Minor Version Number* | 00.13 | 578 |
| *VfsGetFileAttributes* | 492 | *Vfs Get File Status* | 00.21 | 582 |
| *VfsGetFileDefaultMarkIn* | 493 | *Vfs Get File Status* | 00.21 | 582 |
| *VfsGetFileDefaultMarkOut* | 494 | *Vfs Get File Status* | 00.21 | 582 |
| *VfsGetFileDefaultMarks* | 495 | *Vfs Get File Status* | 00.21 | 582 |
| *VfsGetFileModificationTime* | 496 | *Vfs Get File Status* | 00.21 | 582 |
| *VfsGetFileRecordedDuration* | 497 | *Vfs Get File Status* | 00.21 | 582 |
| *VfsGetFileSize* | 498 | *Vfs Get File Size* | 00.41 | 594 |
| *VfsGetFileTimeCode* | 499 | *Vfs Get File Time Code* | 00.40 | 594 |
| *VfsGetFileType* | 500 | *Vfs Get File Status* | 00.21 | 582 |
| *VfsGetFileVideoFormat* | 501 | *Vfs Get File Status* | 00.21 | 582 |
| *VfsGetMajorVersion* | 502 | *Get TekVfs Major Version Number* | 00.1D | 580 |
| *VfsGetMinorVersion* | 503 | *Get TekVfs Minor Version Number* | 00.1E | 581 |
| *VfsMoveFile* | 505 | *Vfs Move File* | 00.25 | 584 |
| *VfsMoveMediaSegment* | 506 | *Vfs Move Media Segment* | 00.30 | 589 |
| *VfsQueryFileSystemSpace* | 507 | *Vfs Query File System Space* | 00.3F | 594 |
| *VfsReadFile* | 508 | *Vfs Read File* | 00.26 | 585 |

**Table 6. Vfs library commands (Continued)**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| | | *Vfs Query File System Space* | 00.3F | 594 |
| *VfsSetCurrentDirectory* | 509 | *Set Current Directory* | 00.0E | 577 |
| *VfsSetFileAttributes* | 510 | *Vfs Set File Attributes* | 00.27 | 585 |
| *VfsStatusOfCopy* | 511 | *Vfs Status of Copy* | 00.31 | 589 |
| *VfsStripeTimecode* | 512 | *Vfs Stripe Timecode* | 00.39 | 592 |
| *VfsWriteFile* | 513 | *Vfs Write File* | 00.28 | 586 |

**Table 7. Vme library commands**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *VmeAllocateContext* | 514 | *Allocate Context* | 80.00 | 750 |
| *VmeApplyContext* | 515 | *Apply Context* | 80.02 | 750 |
| *VmeFreeContext* | 516 | *Free Context* | 80.01 | 750 |
| *VmeGetAspect* | 517 | *Get Position Settings* | 80.21 | 764 |
| *VmeGetBorderColor* | 518 | *Get Border and Edge Settings* | 80.22 | 764 |
| *VmeGetBorderColorYUV* | 519 | *No serial equivalent.* | | |
| *VmeGetBorderWidth* | 520 | *Get Border and Edge Settings* | 80.22 | 764 |
| *VmeGetEdgeSoftness* | 521 | *Get Border and Edge Settings* | 80.22 | 764 |
| *VmeGetFadeLength* | 522 | *Get Transition and Fade Settings* | 80.20 | 763 |
| *VmeGetFadeMatteColor* | 523 | *Get Fade Matte Color Settings* | 80.26 | 766 |
| *VmeGetFadeProgress* | 524 | *Get Fade Progress* | 80.0B | 755 |
| *VmeGetHModulation* | 525 | *Get Horizontal Modulation Settings* | 80.23 | 765 |
| *VmeGetKey* | 526 | *Get Key Settings* | 80.25 | 766 |
| *VmeGetMajorVersion* | 527 | *Get TekVme Major Version Number* | 80.1D | 763 |
| *VmeGetMinorVersion* | 528 | *Get TekVme Minor Version Number* | 80.1E | 763 |
| *VmeGetMultipliers* | 529 | *Get Position Settings* | 80.21 | 764 |
| *VmeGetPosition* | 530 | *Get Position Settings* | 80.21 | 764 |
| *VmeGetRotation* | 531 | *Get Position Settings* | 80.21 | 764 |
| *VmeGetTransitionLength* | 532 | *Get Transition and Fade Settings* | 80.20 | 763 |
| *VmeGetTransitionProgress* | 533 | *Get Transition Progress* | 80.07 | 753 |
| *VmeGetTransitionType* | 534 | *Get Transition Type* | 80.1F | 763 |
| *VmeGetVModulation* | 535 | *Get Vertical Modulation Settings* | 80.24 | 765 |
| *VmeResetFade* | 536 | *Reset Fade* | 80.08 | 753 |
| *VmeResetTransition* | 537 | *Reset Transition Type* | 80.04 | 751 |
| *VmeSetAspect* | 538 | *Set Aspect (not fully implemented)* | 80.11 | 757 |
| *VmeSetBorderColor* | 539 | *Set Border Color* | 80.14 | 759 |
| *VmeSetBorderColorYUV* | 540 | *No serial equivalent.* | | |
| *VmeSetBorderWidth* | 541 | *Set Border Width* | 80.15 | 759 |
| *VmeSetEdgeSoftness* | 542 | *Set Edge Softness* | 80.16 | 760 |
| *VmeSetFadeLength* | 543 | *Set Fade Length* | 80.0E | 756 |
| *VmeSetFadeMatteColor* | 544 | *Set Fade Matte Color* | 80.1A | 762 |
| *VmeSetFadeProgress* | 545 | *Set Fade Progress* | 80.0A | 754 |
| *VmeSetHModulation* | 546 | *Set Horizontal Modulation* | 80.17 | 760 |
| *VmeSetKey* | 547 | *Set Key* | 80.19 | 762 |
| *VmeSetMultipliers* | 548 | *Set Multipliers* | 80.10 | 757 |
| *VmeSetPosition* | 549 | *Set Position* | 80.12 | 758 |
| *VmeSetRotation* | 550 | *Set Rotation* | 80.13 | 758 |
| *VmeSetTransitionLength* | 551 | *Set Transition Length* | 80.0C | 755 |
| *VmeSetTransitionProgress* | 552 | *Set Transition Progress* | 80.06 | 752 |

**Table 7. Vme library commands (Continued)**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *VmeSetTransitionType* | 553 | *Set Transition Type* | 80.03 | 751 |
| *VmeSetVModulation* | 554 | *Set Vertical Modulation* | 80.18 | 761 |
| *VmeStartFade* | 555 | *Start Fade* | 80.09 | 754 |
| *VmeStartFadeAtGenTc* | 556 | *Start Fade At Gen Tc* | 80.1B | 763 |
| *VmeStartTransition* | 557 | *Start Transition* | 80.05 | 752 |
| *VmeStartTransitionAtGenTc* | 558 | *Start Transition At Gen Tc*763 | 80.1C | 763 |

**Table 8. Xfr library commands**

| API command name | Page | Serial equivalent | Command | Page |
|---|---|---|---|---|
| *XfrAbort* | 559 | *Xfr Abort* | 90.02 | 768 |
| *XfrGetActiveTokens* | 560 | *Xfr Get Active Tokens* | 90.03 | 769 |
| *XfrGetStatus* | 561 | *Xfr Get Status* | 90.01 | 768 |
| *XfrRequest* | 562 | *Xfr Request* | 90.00 | 767 |

# Serial command summary

**Table 9. Serial commands: system access**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 00.00 | *Get Transaction Reply* | 573 | *No API equivalent.* | |
| 00.01 | *Get Protocol Version* | 573 | *No API equivalent.* | |
| 00.02 | *Get Device ID* | 574 | *No API equivalent.* | |
| 00.03 | *Get Disk Capacity*<br>bitmask for the following calls: | 574 | *VdrGetRecordCapacity* | 386 |
| | | | *VdrGetTotalRecordCapacity* | 395 |
| 00.04 | *Get Number of File Systems* | 574 | *CfgGetNumFileSystems* | 73 |
| 00.05 | *Get File System Name* | 575 | *CfgGetFileSystemName* | 66 |
| 00.06 | *Get Number of Types*<br>bitmap: | 575 | *CfgGetNumCodecs* | 72 |
| | | | *CfgGetNumInputs* | 74 |
| | | | *CfgGetNumOutputs* | 75 |
| | | | *CfgGetNumTCGenerators* | 77 |
| | | | *CfgGetNumTCRecorders* | 78 |
| 00.07 | *Get Input Name* | 575 | *CfgGetInputName* | 68 |
| 00.08 | *Get Output Name* | 576 | *CfgGetOutputName* | 80 |
| 00.09 | *Get Input Unit* | 576 | *CfgGetInputUnitNum* | 69 |
| 00.0A | *Get Output Unit* | 576 | *CfgGetOutputUnitNum* | 81 |
| 00.0B | *Get Standard* | 576 | *CfgGetStandard* | 83 |
| 00.0C | *Get Video Standard* | 576 | *CfgGetVideoInputStandard* | 84 |
| 00.0D | *Get Current Directory* | 577 | *VfsGetCurrentDirectory* | 488 |
| 00.0E | *Set Current Directory* | 577 | *VfsSetCurrentDirectory* | 509 |
| 00.0F | *Get Current Drive* | 577 | *VfsGetCurrentDrive* | 489 |
| 00.10 | *Get TekVdr Major Version Number* | 577 | *VdrGetMajorVersion* | 335 |
| 00.11 | *Get TekVdr Minor Version Number* | 577 | *VdrGetMinorVersion* | 351 |
| 00.12 | *Get Engine Major Version Number* | 578 | *VfsGetEngineMajorVersion* | 490 |
| 00.13 | *Get Engine Minor Version Number* | 578 | *VfsGetEngineMinorVersion* | 491 |
| 00.14 | *Get Video Format* | 578 | *VdrGetVideoFormat* | 399 |
| 00.15 | *Set Video Format* | 578 | *VdrSetVideoFormat* | 457 |
| 00.16 | *Get TekCfg Major Version Number* | 578 | *CfgGetMajorVersion* | 70 |
| 00.17 | *Get TekCfg Minor Version Number* | 579 | *CfgGetMinorVersion* | 71 |
| 00.18 | *Get Attributes by Type*<br>bitmap: | 579 | *CfgGetCodecAttributes* | 64 |
| | | | *CfgGetInputAttributes* | 67 |
| | | | *CfgGetOutputAttributes* | 79 |
| 00.19 | *Get Video Field Count* | 580 | *VdrGetVideoFieldCount* | 398 |
| 00.1A | *Wait For Video Field* | 580 | *VdrWaitForVideoField* | 469 |
| 00.1B | *Get TekPdr Major Version Number* | 580 | *PdrGetMajorVersion* | 121 |
| 00.1C | *Get TekPdr Minor Version Number* | 580 | *PdrGetMinorVersion* | 135 |

**Table 9. Serial commands: system access (Continued)**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 00.1D | *Get TekVfs Major Version Number* | 580 | *VfsGetMajorVersion* | 502 |
| 00.1E | *Get TekVfs Minor Version Number* | 581 | *VfsGetMinorVersion* | 503 |
| 00.1F | *Directory Exists* | 581 | *VfsDirectoryExists* | 479 |
| 00.20 | *Create Directory* | 581 | *VfsCreateDirectory* | 475 |
| 00.21 | *Vfs Get File Status*<br>bitmap: | 582 | *VfsFileExists* | 484 |
| | | | *VfsGetFileAttributes* | 492 |
| | | | *VfsGetFileDefaultMarkIn* | 493 |
| | | | *VfsGetFileDefaultMarkOut* | 494 |
| | | | *VfsGetFileDefaultMarks* | 495 |
| | | | *VfsGetFileModificationTime* | 496 |
| | | | *VfsGetFileRecordedDuration* | 497 |
| | | | *VfsGetFileType* | 500 |
| | | | *VfsGetFileVideoFormat* | 501 |
| 00.22 | *Vfs Create Media File* | 583 | *VfsCreateMediaFile* | 477 |
| 00.23 | *Vfs Create File* | 583 | *VfsCreateFile* | 476 |
| 00.24 | *Vfs Delete File* | 584 | *VfsDeleteFile* | 478 |
| 00.25 | *Vfs Move File* | 584 | *VfsMoveFile* | 505 |
| 00.26 | *Vfs Read File* | 585 | *VfsReadFile* | 508 |
| 00.27 | *Vfs Set File Attributes* | 585 | *VfsSetFileAttributes* | 510 |
| 00.28 | *Vfs Write File* | 586 | *VfsWriteFile* | 513 |
| 00.29 | *Vfs Erase Media Segment* | 586 | *VfsEraseMediaSegment* | 480 |
| 00.2A | *Vfs Close Handle* | 587 | *VfsCloseHandle* | 472 |
| 00.2B | *Vfs Find First File* | 587 | *VfsFindFirstFile* | 486 |
| 00.2C | *Vfs Find Next File* | 587 | *VfsFindNextFile* | 487 |
| 00.2D | *Vfs Find Close* | 588 | *VfsFindClose* | 485 |
| 00.2E | *Vfs Copy File* | 588 | *VfsCopyFile* | 473 |
| 00.2F | *Vfs Copy Media Segment* | 588 | *VfsCopyMediaSegment* | 474 |
| 00.30 | *Vfs Move Media Segment* | 589 | *VfsMoveMediaSegment* | 506 |
| 00.31 | *Vfs Status of Copy* | 589 | *VfsStatusOfCopy* | 511 |
| 00.32 | *Vfs Cancel Copy* | 590 | *VfsCancelCopy* | 471 |
| 00.33 | *Vfs Field Read* | 590 | *VfsFieldRead* | 482 |
| 00.34 | *Vfs Field Read Data* | 591 | *VfsFieldRead* | 482 |
| 00.35 | *Vfs Field Free* | 591 | *VfsFieldFree* | 481 |
| 00.36 | *Vfs Field Write Data* | 591 | *VfsFieldWrite* | 483 |
| 00.37 | *Vfs Field Write Commit* | 592 | *VfsFieldWrite* | 483 |
| 00.38 | *Get Default Codec Timing* | 592 | *CfgGetCodecTimingMode* | 65 |
| 00.39 | *Vfs Stripe Timecode* | 592 | *VfsStripeTimecode* | 512 |
| 00.3A | *Vfs Set Default Codec Timing (disabled)* | 593 | *No API equivalent.* | |
| 00.3B | *Vfs Set Time* | 593 | *No API equivalent.* | |

**Table 9. Serial commands: system access (Continued)**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 00.3C | *Vfs Get Time* | 593 | *No API equivalent.* | |
| 00.3D | *Get Resource Name* | 593 | *CfgGetResourceName* | 82 |
| 00.3E | *Get Number of Resources* | 593 | *CfgGetNumResources* | 76 |
| 00.3F | *Vfs Query File System Space* | 594 | *VfsQueryFileSystemSpace* | 507 |
| 00.40 | *Vfs Get File Time Code* | 594 | *VfsGetFileTimeCode* | 499 |
| 00.41 | *Vfs Get File Size* | 594 | *VfsGetFileSize* | 498 |
| 00.FF | *Event Poll* | 594 | *PdrCreateEvent* | 93 |

**Table 10. Serial commands: PLS access**

| Command | Serial equivalent | Page | API command name | Page |
|---|---|---|---|---|
| 0E.00 | *Create Extension* | 597 | *No API equivalent.* | |
| 0E.01 | *PlsGetMajorVersion* | 597 | *PlsGetMajorVersion* | 244 |
| 0E.02 | *PlsGetMinorVersion* | 597 | *PlsGetMinorVersion* | 245 |
| 0E.03 | *PlsOpenLibrary* | 598 | *PlsOpenLibrary* | 258 |
| 0E.04 | *PlsCloseLibrary* | 598 | *PlsCloseLibrary* | 209 |
| 0E.05 | *PlsGetLibraryConfig* | 599 | *PlsGetLibraryConfig* | 241 |
| 0E.06 | *PlsGetLibraryStatus* | 600 | *PlsGetLibraryStatus* | 242 |
| 0E.07 | *PlsFindFirstTransportConfig* | 601 | *PlsFindFirstTransportConfig* | 224 |
| 0E.08 | *PlsFindNextTransportConfig* | 603 | *PlsFindNextTransportConfig* | 229 |
| 0E.09 | *PlsCloseTransportConfig* | 603 | *PlsCloseTransportConfig* | 211 |
| 0E.0A | *PlsFindFirstBinInfo* | 604 | *PlsFindFirstBinInfo* | 218 |
| 0E.0B | *PlsFindNextBinInfo* | 605 | *PlsFindNextBinInfo* | 225 |
| 0E.0C | *PlsCloseBinInfo* | 605 | *PlsCloseBinInfo* | 203 |
| 0E.0D | *PlsFindFirstCartridgeInfo* | 606 | *PlsFindFirstCartridgeInfo* | 219 |
| 0E.0E | *PlsFindNextCartridgeInfo* | 608 | *PlsFindNextCartridgeInfo* | 226 |
| 0E.0F | *PlsCloseCartridgeInfo* | 609 | *PlsCloseCartridgeInfo* | 205 |
| 0E.10 | *PlsFindFirstFileInfo* | 609 | *PlsFindFirstFileInfo* | 221 |
| 0E.11 | *PlsFindNextFileInfo* | 610 | *PlsFindNextFileInfo* | 227 |
| 0E.12 | *PlsCloseFileInfo* | 611 | *PlsCloseFileInfo* | 207 |
| 0E.13 | *PlsFindFirstHandle* | 611 | *PlsFindFirstHandle* | 223 |
| 0E.14 | *PlsFindNextHandle* | 613 | *PlsFindNextHandle* | 228 |
| 0E.15 | *PlsCloseHandle* | 613 | *PlsCloseFindHandle* | 208 |
| 0E.20 | *PlsConnectTransport* | 614 | *PlsConnectTransport* | 214 |
| 0E.21 | *PlsAllocateTransport* | 614 | *PlsAllocateTransport* | 197 |
| 0E.22 | *PlsCloseTransport* | 614 | *PlsCloseTransport* | 210 |
| 0E.23 | *PlsGetTransportStatus* | 615 | *PlsGetTransportStatus* | 251 |
| 0E.24 | *PlsLoadTransport* | 617 | *PlsLoadTransport* | 257 |
| 0E.25 | *PlsUnloadTransport* | 617 | *PlsUnloadTransport* | 270 |
| 0E.26 | *PlsCleanTransport* | 617 | *PlsCleanTransport* | 202 |
| 0E.30 | *PlsConnectCartridge* | 618 | *PlsConnectCartridge* | 212 |
| 0E.32 | *PlsAllocateCartridge* | 618 | *PlsAllocateCartridge* | 196 |
| 0E.33 | *PlsCloseCartridge* | 618 | *PlsCloseCartridge* | 204 |
| 0E.34 | *PlsGetCartridgeConfig* | 619 | *PlsGetCartridgeConfig* | 235 |
| 0E.35 | *PlsGetCartridgeStatus* | 620 | *PlsGetCartridgeStatus* | 236 |
| 0E.36 | *PlsInventoryCartridge* | 621 | *PlsInventoryCartridge* | 255 |
| 0E.37 | *PlsUpdateCartridge* | 621 | *PlsUpdateCartridge* | 271 |
| 0E.38 | *PlsFormatCartridge* | 621 | *PlsFormatCartridge* | 230 |
| 0E.3A | *PlsCopyCartridge* | 622 | *PlsCopyCartridge* | 215 |
| 0E.3C | *PlsExportCartridge* | 622 | *PlsExportCartridge* | 217 |

**Table 10. Serial commands: PLS access (Continued)**

| Command | Serial equivalent | Page | API command name | Page |
|---|---|---|---|---|
| 0E.3D | *PlsImportCartridge* | 622 | *PlsImportCartridge* | 253 |
| 0E.3E | *PlsGetLocationString* | 623 | *PlsGetLocationString* | 243 |
| 0E.3F | *PlsSetLocationString* | 623 | *PlsSetLocationString* | 267 |
| 0E.40 | *PlsSetCartDescription* | 623 | *PlsSetCartDescription* | 264 |
| 0E.41 | *PlsGetCartDescription* | 624 | *PlsGetCartDescription* | 234 |
| 0E.42 | *PlsImportLoadCartridge* | 624 | *PlsImportLoadCartridge* | 254 |
| 0E.43 | *PlsGetPartitionMap* | 624 | *PlsGetPartitionMap* | 247 |
| 0E.50 | *PlsConnectFile* | 625 | *PlsConnectFile* | 213 |
| 0E.51 | *PlsCloseFile* | 625 | *PlsCloseFile* | 206 |
| 0E.53 | *PlsArchiveClip* | 625 | *PlsArchiveClip* | 198 |
| 0E.54 | *PlsArchiveDataFile* | 626 | *PlsArchiveDataFile* | 199 |
| 0E.56 | *PlsRestore* | 626 | *PlsRestore* | 261 |
| 0E.57 | *PlsRenameFile* | 626 | *PlsRenameFile* | 260 |
| 0E.58 | *PlsDeleteFile* | 627 | *PlsDeleteFile* | 216 |
| 0E.5A | *PlsSetFileDescription* | 627 | *PlsSetFileDescription* | 266 |
| 0E.5B | *PlsGetFileDescription* | 627 | *PlsGetFileDescription* | 240 |
| 0E.5F | *PlsRestoreDataFile* | 628 | *PlsRestoreDataFile* | 262 |
| 0E.60 | *PlsSetBackupDirCommand* | 628 | *PlsSetBackupDir* | 263 |
| 0E.61 | *PlsGetBackupDirCommand* | 628 | *PlsGetBackupDir* | 233 |
| 0E.62 | *PlsBackupCatalogCommand* | 629 | *PlsBackupCatalog* | 200 |
| 0E.63 | *PlsGetClipSize* | 629 | *PlsGetClipSize* | 237 |
| 0E.70 | *PlsSetEventMask* | 629 | *PlsSetEventMask* | 265 |
| 0E.71 | *PlsGetEventMask* | 630 | *PlsGetEventMask* | 239 |
| 0E.72 | *PlsGetAnyEvent* | 630 | *PlsGetAnyEvent* | 231 |
| 0E.73 | *PlsGetAsynchEvent* | 634 | *PlsGetAsynchEvent* | 232 |
| 0E.74 | *PlsGetCommandEvent* | 638 | *PlsGetCommandEvent* | 238 |
| 0E.75 | *PlsSetModes* | 642 | *PlsSetModes* | 268 |
| 0E.76 | *PlsGetModes* | 642 | *PlsGetModes* | 246 |
| 0E.77 | *PlsSetPath* | 642 | *PlsSetPath* | 269 |
| 0E.78 | *PlsGetPath* | 643 | *PlsGetPath* | 248 |
| 0E.79 | *PlsGetTimeDate* | 643 | *PlsGetTimeDate* | 250 |
| 0E.7A | *PlsAddTransport* | 643 | *PlsAddTransport* | 195 |
| 0E.7B | *PlsRemoveTransport* | 644 | *PlsRemoveTransport* | 259 |
| 0E.7C | *PlsInventoryLibrary* | 644 | *PlsInventoryLibrary* | 256 |
| 0E.7D | *PlsGetStatusCommand* | 644 | *PlsGetStatusCommand* | 249 |
| 0E.7E | *PlsCancelCommand* | 645 | *PlsCancelCommand* | 201 |
| 0E.7F | *PlsHouseKeeping* | 645 | *PlsHouseKeeping* | 252 |
| 0E.FF | *Delete Extension* | 645 | *No API equivalent.* | |
| 0E.xx | *Extension Command Execution* | 646 | *No API equivalent.* | |

**Table 11. Serial commands: port access**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 10.00 | *Get Port Status* bitmap: | 647 | *VdrGetCurrentRate* | 327 |
| | | | *VdrGetCurrentState* | 328 |
| | | | *VdrGetFieldDominance* | 332 |
| | | | *VdrGetPlayMode* | 374 |
| | | | *VdrGetPosition* | 379 |
| | | | *VdrGetMovieAtPosition* | 354 |
| | | | *VdrGetStillPlayMode* | 394 |
| 10.01 | *Open Port* | 649 | *VdrOpenPortConnection* | 408 |
| 10.02 | *Close Port* | 649 | *VdrClosePort* | 300 |
| 10.03 | *Idle* | 650 | *VdrIdle* | 401 |
| 10.04 | *Cue Play* | 650 | *VdrCuePlay* | 301 |
| 10.05 | *Shuttle Play* | 651 | *VdrShuttle* | 459 |
| 10.06 | *Jog Play* | 651 | *VdrJog* | 406 |
| 10.07 | *Cue Record* | 652 | *VdrCueRecord* | 302 |
| 10.08 | *Shuttle Record* | 652 | *VdrShuttle* | 459 |
| 10.09 | *Jog Record* | 653 | *VdrJog* | 406 |
| 10.0A | *Set Position* | 653 | *VdrSetPosition* | 450 |
| | | | *VdrSetPositionRelative* | 452 |
| 10.0B | *Set Motion Play/Record Mode* | 654 | *VdrSetPlayMode* | 448 |
| 10.0C | *Set Still Play Mode* | 655 | *VdrSetStillPlayMode* | 455 |
| 10.0D | *Set Position Limits* | 655 | *VdrSetMaxPosition* | 434 |
| | | | *VdrSetMinPosition* | 442 |
| | | | *VdrSetPositionLimits* | 451 |
| 10.0E | *Get Position Limits* | 656 | *VdrGetMaxPosition* | 338 |
| | | | *VdrGetMinPosition* | 352 |
| | | | *VdrGetPosition* | 379 |
| 10.0F | *Get Next Open Port* | 656 | *VdrGetNextPort* | 370 |
| 10.10 | *Get Previous Open Port* | 656 | *VdrGetPreviousPort* | 383 |
| 10.11 | *Set Field Dominance* | 657 | *VdrSetFieldDominance* | 423 |
| 10.12 | *Reset Port* | 657 | *VdrResetPort* | 411 |
| 10.13 | *Set Default Tc Format* | 657 | *VdrSetDefaultTcFormat* | 421 |
| 10.14 | *Shuttle At Gen Tc* | 657 | *VdrShuttleAtGenTc* | 460 |
| 10.15 | *Cancel At Gen Tc Cmds* | 657 | *VdrCancelPendingTimecodeTrigger-Commands* | 299 |

**Table 12. Serial commands: PDR movie access**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 21.00 | *Pdr Get Movie Status* bitmap: | 659 | *PdrGetMovieAttributes* | 136 |
| | | | *PdrGetMovieCreateTime* | 139 |
| | | | *PdrGetMovieDataset* | 140 |
| | | | *PdrGetMovieGroup* | 141 |
| | | | *PdrGetMovieLastChangeTime* | 142 |
| | | | *PdrGetMovieLength* | 143 |
| | | | *PdrGetMovieName* | 146 |
| 21.01 | *Pdr Find First Dataset* | 660 | *PdrFindFirstDataset* | 101 |
| 21.02 | *Pdr Find Next Dataset* | 660 | *PdrFindNextDataset* | 106 |
| 21.03 | *Pdr Close Find* | 660 | *PdrCloseFind* | 87 |
| 21.04 | *Pdr Get Current Dataset* | 661 | *PdrGetCurrentDataset* | 112 |
| 21.05 | *Pdr Set Current Dataset* | 661 | *PdrSetCurrentDataset* | 178 |
| 21.06 | *Pdr Find First Group* | 661 | *PdrFindFirstGroup* | 103 |
| 21.07 | *Pdr Find Next Group* | 662 | *PdrFindNextGroup* | 108 |
| 21.08 | *Pdr Get Current Group* | 662 | *PdrGetCurrentGroup* | 113 |
| 21.09 | *Pdr Set Current Group* | 662 | *PdrSetCurrentGroup* | 179 |
| 21.0A | *Pdr Find First Movie* | 663 | *PdrFindFirstMovie* | 105 |
| 21.0B | *Pdr Find Next Movie* | 663 | *PdrFindNextMovie* | 110 |
| 21.0C | *Pdr Create Movie* | 663 | *PdrCreateMovie* | 96 |
| 21.0D | *Pdr Open Movie* | 664 | *PdrOpenMovie* | 170 |
| 21.0E | *Pdr Close Movie* | 664 | *PdrCloseMovie* | 88 |
| 21.0F | *Pdr Copy Movie Token* | 664 | *PdrCopyMovieToken* | 91 |
| 21.10 | *Pdr Rename Movie* | 665 | *PdrRenameMovie* | 172 |
| 21.11 | *Pdr Delete Movie* | 665 | *PdrDeleteMovie* | 100 |
| 21.12 | *Pdr Get Movie State* | 666 | *PdrGetMovieStateInfo* | 147 |
| 21.13 | *Pdr Set Movie Attributes* bitmap: | 666 | *PdrSetMovieLocked* | 183 |
| | | | *PdrSetMovieReadOnly* | 187 |
| | | | *PdrSetMovieReadWrite* | 188 |
| | | | *PdrSetMovieUnlocked* | 189 |
| 21.14 | *Pdr Get Number of Movies* | 667 | *PdrGetNumMovies* | 152 |
| 21.15 | *Pdr Movie Exists* | 667 | *PdrMovieExists* | 168 |
| 21.16 | *Pdr Set Clip Marks* bitmap: | 667 | *PdrSetClipIn* | 176 |
| | | | *PdrSetClipOut* | 177 |
| 21.17 | *Pdr Trim Clip* | 668 | *PdrTrimClip* | 194 |
| 21.18 | *Pdr Copy Movie* | 668 | *PdrCopyMovie* | 90 |
| 21.19 | *Pdr Save Movie* | 669 | *PdrSaveMovie* | 175 |
| 21.1A | *Pdr Get Wait Operation Status* | 669 | *PdrGetWaitOpStatus* | 165 |
| 21.1B | *Pdr Terminate Wait Operation* | 670 | *PdrTerminateWaitOperation* | 193 |
| 21.1C | *Pdr Set User Data* | 670 | *PdrSetUserData* | 192 |

**Table 12. Serial commands: PDR movie access (Continued)**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 21.1D | *Pdr Set User Data Commit* | 671 | *PdrSetUserData* | 192 |
| 21.1E | *Pdr Get User Data* | 671 | *PdrGetUserData* | 164 |
| 21.1F | *Pdr Get User Data Commit* | 672 | *PdrGetUserData* | 164 |
| 21.20 | *Pdr Backup Movie Descriptor* | 672 | *PdrBackupMovieDescription* | 85 |
| 21.21 | *Pdr Restore Movie Descriptor* | 673 | *PdrRestoreMovieDescription* | 174 |
| 21.22 | *Pdr Get List of Names* | 673 | *PdrGetListOfNames* | 119 |
| 21.23 | *Pdr Get Movie Changes* | 674 | *PdrGetMovieChanges* | 137 |
| 21.24 | *Pdr Create Event* | 674 | *PdrCreateEvent* | 93 |
| 21.25 | *Pdr Set Registry* | 675 | *PdrSetRegistry* | 190 |
| 21.26 | *Pdr Get Registry* | 675 | *PdrGetRegistry* | 157 |
| 21.27 | *Get First Timecode Position* | 676 | *PdrGetFirstTcPosition* | 116 |
| 21.28 | *Get Last Timecode Position* | 676 | *PdrGetLastTcPosition* | 118 |
| 21.29 | *Rename Open Movie* | 676 | *PdrRenameOpenMovie* | 173 |
| 21.2A | *(Reserved)* | 677 | *No API equivalent.* | |
| 21.2B | *Movie To Media Name* | 677 | *No API equivalent.* | |
| 21.2C | *Pdr Set Small User Data* | 678 | *No API equivalent.* | |
| 21.2D | *Pdr Get Small User Data* | 678 | *No API equivalent.* | |
| 21.2E | *Set Open Movie Marks*<br>bitmap: | 679 | *PdrSetMovieMarkIn* | 184 |
| | | | *PdrSetMovieMarkOut* | 185 |
| 21.2F | *Set Movie Marks* | 679 | *PdrSetMovieMarks* | 186 |
| 21.30 | *Pdr Save Movie By Port* | 679 | *PdrSaveMovie* | 175 |
| 21.31 | *Pdr Close All Enumerators* | 680 | *PdrCloseAllEnumTokens* | 86 |
| 21.32 | *Pdr Get Movie State Info* | 681 | *PdrGetMovieStateInfo* | 147 |

**Table 13. Serial commands: attached movie access**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 30.00 | *Get Attached Movie Status* bitmap: | 682 | *VdrGetMovieAccessMode* | 353 |
| | | | *VdrGetMovieDefaultMarkIn* | 355 |
| | | | *VdrGetMovieDefaultMarkOut* | 356 |
| | | | *VdrGetMovieEndPosition* | 357 |
| | | | *VdrGetMovieMarkDuration* | 358 |
| | | | *VdrGetMovieMarkIn* | 359 |
| | | | *VdrGetMovieMarkOut* | 360 |
| | | | *VdrGetMovieName* | 361 |
| | | | *VdrGetMovieStartPosition* | 362 |
| | | | *VdrGetMovieUserData* | 363 |
| 30.01 | *Attach Movie* | 684 | *VdrAttachMovie* | 291 |
| 30.02 | *Attach Movie with Marks* | 685 | *VdrAttachMovieWithMarks* | 293 |
| 30.03 | *Detach Movie* | 686 | *VdrDetachMovie* | 310 |
| 30.04 | *Get Next Movie* | 686 | *VdrGetNextMovie* | 369 |
| 30.05 | *Get Previous Movie* | 687 | *VdrGetPreviousMovie* | 382 |
| 30.06 | *Get Movie at Position* | 687 | *VdrGetMovieAtPosition* | 354 |
| 30.07 | *Set Movie Mark In* | 688 | *VdrSetMovieMarkIn* | 443 |
| 30.08 | *Set Movie Mark Out* | 689 | *VdrSetMovieMarkOut* | 444 |
| 30.09 | *Get Port Handle from Movie Handle* | 689 | *VdrGetPortFromMovie* | 377 |
| 30.0A | *Set Movie User Data* | 690 | *VdrSetMovieUserData* | 445 |
| 30.0B | *Attach Movie With Resources* | 690 | *VdrAttachMovie* | 291 |
| 30.0C | *Expand Movie* | 691 | *VdrExpandMovie* | 313 |
| 30.0D | *Get Movie at Resource Position* | 691 | *VdrGetMovieAtPosition* | 354 |
| 30.0E | *Attach Open Movie* | 692 | *VdrAttachOpenMovie* | 295 |
| 30.0F | *Disable Movie Position Limits* | 692 | *VdrDisableMoviePositionLimiting* | 311 |
| 30.10 | *Enable Movie Position Limits* | 693 | *VdrEnableMoviePositionLimiting* | 312 |
| 30.11 | *Is Movie Position Limiting Enabled* | 693 | *VdrIsMoviePositionLimitingEnabled* | 403 |
| 30.12 | *Attach Movie With Marks* | 694 | *VdrAttachMovieWithMarks* | 293 |
| 30.13 | *Attach Open Movie With Marks* | 695 | *VdrAttachOpenMovieWithMarks* | 296 |
| 30.14 | *Expand Movie By Port* | 696 | *VdrExpandMovie* | 313 |
| 30.15 | *Attach Open Movie By Port* | 697 | *VdrAttachOpenMovie* | 295 |
| 30.16 | *Attach Open Movie With Marks By Port* | 698 | *VdrAttachOpenMovieWithMarks* | 296 |

**Table 14. Serial commands: stored media access**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 40.00 | *Get Stored Media Status* | 699 | *PdrGetMediaAttributes* | 123 |
| | | | *PdrGetMediaDataset* | 124 |
| | | | *PdrGetMediaDataType* | 125 |
| | | | *PdrGetMediaDuration* | 126 |
| | | | *PdrGetMediaIn* | 127 |
| | | | *PdrGetMediaMarks* | 128 |
| | | | *PdrGetMediaOut* | 129 |
| | | | *PdrGetMediaPath* | 130 |
| | | | *PdrGetMediaTokenType* | 134 |
| | | | *PdrGetMovieTokenFromMediaToken* | 148 |
| | | | *PdrGetPositionOfMediaToken* | 154 |
| 40.01 | *Get Maximum Number of Tracks* | 700 | *PdrGetMaxTracks* | 122 |
| 40.02 | *Get Number of Tracks* | 701 | *PdrGetNumTracks* | 153 |
| 40.03 | *Get Next Track* | 701 | *PdrGetNextTrack* | 150 |
| 40.04 | *Get Previous Track* | 701 | *PdrGetPreviousTrack* | 156 |
| 40.05 | *Get Track Length* | 701 | *PdrGetTrackLength* | 159 |
| 40.06 | *Get Track Token From Media Token* | 702 | *PdrGetTrackTokenFromMediaToken* | 160 |
| 40.07 | *Get Number of Media on Track* | 702 | *PdrGetNumMediaOnTrack* | 151 |
| 40.08 | *Create Media Token* | 702 | *PdrCreateMediaToken* | 94 |
| 40.09 | *Create Black Media Token* | 702 | *PdrCreateBlackMediaToken* | 92 |
| 40.0A | *Create Media Token with Path* | 703 | *PdrCreateMediaTokenWithPath* | 95 |
| 40.0B | *Delete Media* | 703 | *PdrDeleteMedia* | 99 |
| 40.0C | *Copy Media Token* | 703 | *PdrCopyMediaToken* | 89 |
| 40.0D | *Get Next Media Token* | 703 | *PdrGetNextMediaToken* | 149 |
| 40.0E | *Get Previous Media Token* | 704 | *PdrGetPreviousMediaToken* | 155 |
| 40.0F | *Get Media Token at Position* | 704 | *PdrGetMediaTokenAtPosition* | 133 |
| 40.10 | *Set Media Mark In* | 704 | *PdrSetMediaIn* | 180 |
| 40.11 | *Set Media Mark Out* | 704 | *PdrSetMediaOut* | 182 |
| 40.12 | *Set Media Marks* | 705 | *PdrSetMediaMarks* | 181 |
| 40.13 | *Create Track Token* | 705 | *PdrCreateTrackToken* | 97 |
| 40.14 | *Get Track Token Number* | 705 | *PdrGetTrackTokenNum* | 161 |
| 40.15 | *Get Track Token Type* | 706 | *PdrGetTrackTokenType* | 162 |
| 40.16 | *Get Track Type From Resource* | 706 | *PdrGetTrackTypeFromResourceType* | 163 |
| 40.17 | *Get Resource Type From Track Type* | 706 | *PdrGetResourceTypeFromTrackType* | 158 |
| 40.18 | *Get Null Media Token* | 706 | *PdrNullMediaToken* | 169 |
| 40.19 | *Is Media Token Null* | 706 | *PdrMediaTokenIsNull* | 167 |
| 40.1A | *Get Media State* | 707 | *PdrGetMediaStateInfo* | 131 |
| 40.1B | *Get Media State Info* | 708 | *PdrGetMediaStateInfo* | 131 |

**Table 15. Serial commands: attached media access**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 50.00 | *Get Attached Media Status* bitmap: | 710 | *VdrGetMediaDuration* | 341 |
| | | | *VdrGetMediaEndPosition* | 342 |
| | | | *VdrGetMediaMarkDuration* | 343 |
| | | | *VdrGetMediaMarkIn* | 344 |
| | | | *VdrGetMediaMarkOut* | 345 |
| | | | *VdrGetMediaName* | 346 |
| | | | *VdrGetMediaStartPosition* | 347 |
| | | | *VdrGetMediaUserData* | 348 |
| 50.01 | *Attach Media* | 711 | *VdrAttachFittedMedia* | 287 |
| | | | *VdrAttachMedia* | 289 |
| 50.02 | *Attach Media With Marks* | 712 | *VdrAttachFittedMediaWithMarks* | 288 |
| | | | *VdrAttachMediaWithMarks* | 290 |
| 50.03 | *Detach Media* | 713 | *VdrDetachMedia* | 309 |
| 50.04 | *Get Next Media* | 713 | *VdrGetNextMedia* | 368 |
| 50.05 | *Get Previous Media* | 713 | *VdrGetPreviousMedia* | 381 |
| 50.06 | *Get Media at Position* | 714 | *VdrGetMediaAtPosition* | 339 |
| 50.07 | *Set Media Mark In* | 714 | *VdrSetFittedMediaMarkIn* | 424 |
| | | | *VdrSetMediaMarkIn* | 437 |
| 50.08 | *Set Media Mark Out* | 715 | *VdrSetFittedMediaMarkOut* | 425 |
| | | | *VdrSetMediaMarkOut* | 438 |
| 50.09 | *Set Media Duration* | 716 | *VdrSetMediaDuration* | 436 |
| 50.0A | *Get Port Handle From Media Handle* | 716 | *VdrGetPortFromMedia* | 376 |
| 50.0B | *Set Media User Data* | 716 | *VdrSetMediaUserData* | 439 |
| 50.0C | *Get Resource From Media* | 717 | *VdrGetResourceFromMedia* | 388 |
| 50.0D | *Get Media Audio Fades* | 717 | *VdrGetMediaAudioFades* | 340 |
| 50.0E | *Set Media Audio Fades* | 717 | *VdrSetMediaAudioFades* | 435 |
| 50.0F | *Attach Media With Marks And Fades* | 717 | *No API equivalent.* | |
| 50.10 | *Attach Media By Movie With Marks And Fades* | 718 | *No API equivalent.* | |

**Table 16. Serial commands: resource access**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 60.00 | *Get Resource Status* bitmap: | 721 | *VdrGetAudioLevel* | 319 |
| | | | *VdrGetCurrentFieldSize* | 324 |
| | | | *VdrGetCurrentLumQFactor* | 325 |
| | | | *VdrGetCurrentTimeCode* | 329 |
| | | | *VdrGetCurrentUserBits* | 330 |
| 60.01 | *Get Resource Configuration* bitmap: | 723 | *VdrGetAbsMaxChrQ* | 314 |
| | | | *VdrGetAbsMaxLumQ* | 315 |
| | | | *VdrGetAbsMinChrQ* | 316 |
| | | | *VdrGetAbsMinLumQ* | 317 |
| | | | *VdrGetAudioInputLevel* | 318 |
| | | | *VdrGetAudioOutputLevel* | 320 |
| | | | *VdrGetAudioWindow* | 321 |
| | | | *VdrGetBitRate* | 322 |
| | | | *VdrGetGenTcFormat* | 333 |
| | | | *VdrGetGenTcMode* | 334 |
| | | | *VdrGetMaxChrQ* | 336 |
| | | | *VdrGetMaxLumQ* | 337 |
| | | | *VdrGetMinChrQ* | 349 |
| | | | *VdrGetMinLumQ* | 350 |
| | | | *VdrGetMpegChromaFormat* | 364 |
| | | | *VdrGetMpegGopStructure* | 366 |
| | | | *VdrGetResourceNumber* | 389 |
| | | | *VdrGetResourceType* | 390 |
| | | | *VdrGetVideoGoalSize* | 400 |
| | | | *VdrIsResourceActive* | 404 |
| 60.02 | *Allocate Resource* | 726 | *VdrAllocateResource* | 283 |
| 60.03 | *Get Resource Connect Handle* | 728 | *VdrGetResourceConnectionHandle* | 387 |
| 60.04 | *Release Resource* | 729 | *VdrReleaseResource* | 409 |
| 60.05 | *Reset Resource* | 729 | *VdrResetResource* | 412 |
| 60.06 | *Get Next Resource* | 730 | *VdrGetNextResource* | 371 |
| | | | *VdrGetNextResourceOfType* | 372 |
| 60.07 | *Get Previous Resource* | 730 | *VdrGetPreviousResource* | 384 |
| | | | *VdrGetPreviousResourceOfType* | 385 |
| 60.08 | *Activate Resource* | 731 | *VdrActivateResource* | 281 |
| 60.09 | *Deactivate Resource* | 732 | *VdrDeactivateResource* | 303 |
| 60.0A | *Set Video Goal Size* | 733 | *VdrSetVideoGoalSize* | 458 |
| 60.0B | *Set Luminance Q Limits* | 733 | *VdrSetLumQLimits* | 431 |
| | | | *VdrSetMaxLumQ* | 433 |
| | | | *VdrSetMinLumQ* | 441 |

**Table 16. Serial commands: resource access (Continued)**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 60.0C | *Set Chrominance Q Limits* | 734 | *VdrSetChrQLimits* | 420 |
| | | | *VdrSetMaxChrQ* | 432 |
| | | | *VdrSetMinChrQ* | 440 |
| 60.0D | *Set Audio Input Level* | 734 | *VdrSetAudioInputLevel* | 416 |
| 60.0E | *Set Audio Output Level* | 735 | *VdrSetAudioOutputLevel* | 417 |
| 60.0F | *Set Audio Window* | 735 | *VdrSetAudioWindow* | 418 |
| 60.10 | *Set Timecode Generator Format* | 736 | *VdrSetGenTcFormat* | 426 |
| 60.11 | *Set Timecode Generator Mode* | 736 | *VdrSetGenTcMode* | 427 |
| 60.12 | *Is Resource Available* | 737 | *VdrIsResourceAvailable* | 405 |
| 60.13 | *Get Port Handle from Resource Handle* | 737 | *VdrGetPortFromResource* | 378 |
| 60.14 | *Clear Timeline* convenience routine: | 737 | *VdrCancelEvent* | 298 |
| | | | *VdrDetachMedia* | 309 |
| | | | *VdrDetachMovie* | 310 |
| 60.15 | *Set Resource User Data* | 738 | *VdrSetResourceUserData* | 453 |
| 60.16 | *Get Resource User Data* | 738 | *VdrGetResourceUserData* | 391 |
| 60.17 | *Get Video Combiner Part Handle* | 738 | *VdrGetVideoCombinerPartHandle* | 397 |
| 60.18 | *Set Video Codec Timing* | 738 | *VdrSetVideoCodecTiming* | 456 |
| 60.19 | *Set Mpeg Chroma Format* | 739 | *VdrSetMpegChromaFormat* | 446 |
| 60.1A | *Set Mpeg Gop Structure* | 739 | *VdrSetMpegGopStructure* | 447 |
| 60.1B | *Merge Resources* | 739 | *VdrMergeResources* | 407 |
| 60.1C | *Get Abs Port Config* | 739 | *No API equivalent.* | |
| 60.1D | *Set Bit-Rate* | 739 | *VdrSetBitRate* | 419 |
| 60.1E | *Get Mpeg Gop Limits* | 740 | *VdrGetMpegGopLimits* | 365 |
| 60.1F | *Set Encoding Range* | 740 | *VdrSetEncodingRange* | 422 |
| 60.20 | *Set Gen Tc Regen Source* | 740 | *VdrSetGenTcRegenSource* | 428 |

**Table 17. Serial commands: event access**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 70.00 | *Default Event* | 743 | *VdrDefaultEvent* | 304 |
| 70.01 | *Schedule Event* | 744 | *VdrScheduleEvent* | 413 |
| 70.02 | *Reschedule Event* | 745 | *VdrRescheduleEvent* | 410 |
| 70.03 | *(Unused)* | 745 | *No API equivalent.* | |
| 70.04 | *Cancel Event* | 745 | *VdrCancelEvent* | 298 |
| 70.05 | *When Is Event* | 746 | *VdrWhenIsEvent* | 470 |
| 70.06 | *(Reserved)* | 746 | *No API equivalent.* | |
| 70.07 | *(Reserved)* | 746 | *No API equivalent.* | |
| 70.08 | *Get Port Handle from Event Handle* | 746 | *VdrGetPortFromEvent* | 375 |
| 70.09 | *Get Next Event Handle* | 746 | *VdrGetNextEvent* | 367 |
| 70.0A | *Get Previous Event Handle* | 746 | *VdrGetPreviousEvent* | 380 |
| 70.0B | *Default Event At Gen Tc* | 747 | *VdrDefaultEventAtGenTc* | 307 |
| 70.0C | *State Event* | 747 | *VdrStateEvent* | 462 |
| 70.0D | *Default Event At Gen Tc* | 747 | *VdrStateEventAtGenTc* | 463 |

**Table 18. Serial commands: video mix effects access**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 80.00 | *Allocate Context* | 750 | *VmeAllocateContext* | 514 |
| 80.01 | *Free Context* | 750 | *VmeFreeContext* | 516 |
| 80.02 | *Apply Context* | 750 | *VmeApplyContext* | 515 |
| 80.03 | *Set Transition Type* | 751 | *VmeSetTransitionType* | 553 |
| 80.04 | *Reset Transition Type* | 751 | *VmeResetTransition* | 537 |
| 80.05 | *Start Transition* | 752 | *VmeStartTransition* | 557 |
| 80.06 | *Set Transition Progress* | 752 | *VmeSetTransitionProgress* | 552 |
| 80.07 | *Get Transition Progress* | 753 | *VmeGetTransitionProgress* | 533 |
| 80.08 | *Reset Fade* | 753 | *VmeResetFade* | 536 |
| 80.09 | *Start Fade* | 754 | *VmeStartFade* | 555 |
| 80.0A | *Set Fade Progress* | 754 | *VmeSetFadeProgress* | 545 |
| 80.0B | *Get Fade Progress* | 755 | *VmeGetFadeProgress* | 524 |
| 80.0C | *Set Transition Length* | 755 | *VmeSetTransitionLength* | 551 |
| 80.0D | *(Unused)* | 755 | *No API equivalent.* | |
| 80.0E | *Set Fade Length* | 756 | *VmeSetFadeLength* | 543 |
| 80.0F | *(Unused)* | 756 | *No API equivalent.* | |
| 80.10 | *Set Multipliers* | 757 | *VmeSetMultipliers* | 548 |
| 80.11 | *Set Aspect (not fully implemented)* | 757 | *VmeSetAspect* | 538 |
| 80.12 | *Set Position* | 758 | *VmeSetPosition* | 549 |
| 80.13 | *Set Rotation* | 758 | *VmeSetRotation* | 550 |
| 80.14 | *Set Border Color* | 759 | *VmeSetBorderColor* | 539 |
| 80.15 | *Set Border Width* | 759 | *VmeSetBorderWidth* | 541 |
| 80.16 | *Set Edge Softness* | 760 | *VmeSetEdgeSoftness* | 542 |
| 80.17 | *Set Horizontal Modulation* | 760 | *VmeSetHModulation* | 546 |
| 80.18 | *Set Vertical Modulation* | 761 | *VmeSetVModulation* | 554 |
| 80.19 | *Set Key* | 762 | *VmeSetKey* | 547 |
| 80.1A | *Set Fade Matte Color* | 762 | *VmeSetFadeMatteColor* | 544 |
| 80.1B | *Start Fade At Gen Tc* | 763 | *VmeStartFadeAtGenTc* | 556 |
| 80.1C | *Start Transition At Gen Tc* | 763 | *VmeStartTransitionAtGenTc* | 558 |
| 80.1D | *Get TekVme Major Version Number* | 763 | *VmeGetMajorVersion* | 527 |
| 80.1E | *Get TekVme Minor Version Number* | 763 | *VmeGetMinorVersion* | 528 |
| 80.1F | *Get Transition Type* | 763 | *VmeGetTransitionType* | 534 |
| 80.20 | *Get Transition and Fade Settings*<br>bitmap: | 763 | *VmeGetFadeLength* | 522 |
| | | | *VmeGetTransitionLength* | 532 |
| 80.21 | *Get Position Settings*<br>bitmap: | 764 | *VmeGetAspect* | 517 |
| | | | *VmeGetMultipliers* | 529 |
| | | | *VmeGetPosition* | 530 |
| | | | *VmeGetRotation* | 531 |

**Table 18. Serial commands: video mix effects access (Continued)**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 80.22 | *Get Border and Edge Settings*<br>bitmap: | 764 | *VmeGetBorderColor* | 518 |
| | | | *VmeGetBorderWidth* | 520 |
| | | | *VmeGetEdgeSoftness* | 521 |
| 80.23 | *Get Horizontal Modulation Settings* | 765 | *VmeGetHModulation* | 525 |
| 80.24 | *Get Vertical Modulation Settings* | 765 | *VmeGetVModulation* | 535 |
| 80.25 | *Get Key Settings* | 766 | *VmeGetKey* | 526 |
| 80.26 | *Get Fade Matte Color Settings* | 766 | *VmeGetFadeMatteColor* | 523 |

**Table 19. Serial commands: streaming transfer access**

| Command | Serial command name | Page | API equivalent | Page |
|---|---|---|---|---|
| 90.00 | *Xfr Request* | 767 | *XfrRequest* | 562 |
| 90.01 | *Xfr Get Status* | 768 | *XfrGetStatus* | 561 |
| 90.02 | *Xfr Abort* | 768 | *XfrAbort* | 559 |
| 90.03 | *Xfr Get Active Tokens* | 769 | *XfrGetActiveTokens* | 560 |

Chapter 2

# API Commands

This chapter alphabetically lists the functions in the eight Application Programming Interface (API) libraries:


- *The Cfg library*
- *The Pdr library*
- *The Pls library*
- *The Rem library*
- *The Vdr library*
- *The Vfs library*
- *The Vme library*
- *The Xfr library*


For a conceptual overview of the libraries, refer to the *Profile Software Development Kit User Manual.*

# The Cfg library

The TekCfg library provides configuration information for the application.

## CfgGetCodecAttributes

**#include "tekcfg.h"**

**BOOL CfgGetCodecAttributes**(**ConnectHandle** *connHandle*,
**PciCodecType** *codecType*,
**INT** *logicalUnitNumber*,
**PciAttributeTypes_t** *attribute*,
**BOOL*** *result*)

This function returns the attribute of the specified input.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*codecType:* The possible values are:

- AudioCodec
- JpegCodec (same as VideoCodec)
- TimecodeCodec
- MixEffects
- MpegEncoder
- MpegDecoder
- MpegHdDecoder
- MpegHdEncoder
- Dvcpro25Codec
- Dvcpro50Codec

*logicalUnitNumber:* The number of the codec.

*attribute:* This value must be `LevelAdjustment` or `EqualizationAdjustment`. Additional values will be supported in a future release.

*result:* TRUE if the codec has the *attribute*; otherwise, FALSE.

**Return Value:** TRUE if successful; otherwise, FALSE.

## CfgGetCodecTimingMode

**#include "tekcfg.h"**

**Comments:** New command; engineering specification not yet available.

## CfgGetFileSystemName

**#include "tekcfg.h"**

**BOOL CfgGetFileSystemName**(**ConnectHandle** *connHandle*,
**INT** *fileSystemId*,
**CHAR *** *name*,
**INT** *maxLength*)

This function returns the name of the referenced file system.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *fileSystemId:* | The ID for one of the file systems into which the media disk space has been partitioned. This value must be less than the value returned by the function *CfgGetNumFileSystems*. FileSystem 0 always exists, and the disks that are internal to the Profile video server will be assigned to that ID. |
| *name:* | A buffer address into which the name of the file system will be placed. |
| *maxLength:* | The length of the buffer into which the file system name is to be placed. |

**Return Value:** TRUE if the name is placed in the buffer with no problems; otherwise, FALSE.

**See Also:** *CfgGetNumFileSystems* on page 73.

## CfgGetInputAttributes

**#include "tekcfg.h"**

**BOOL CfgGetInputAttributes**(**ConnectHandle** *connHandle*,
**PciInputType** *inputType*,
**INT** *logicalUnitNumber*,
**PciAttributeTypes_t** *attribute*,
**BOOL*** *result*)

This function returns the attribute of the specified input.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*inputType:* This value must be `AudioInput`. Additional values will be supported in a future release.

*logicalUnitNumber:* The number of the audio input.

*attribute:* This value must be `LevelAdjustment` or `EqualizationAdjustment`. Additional values will be supported in a future release.

*result:* TRUE if the input has the *attribute*; otherwise, FALSE.

**Return Value:** TRUE if successful; otherwise, FALSE.

## CfgGetInputName

**#include "tekcfg.h"**

**BOOL CfgGetInputName**(**ConnectHandle** *connHandle*,
**PciInputType** *inputType*,
**INT** *logicalUnitNumber*,
**CHAR *** *name*,
**INT** *maxLength*)

This function returns the name of the requested input.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*inputType:* This value must be one of the following:

- AudioInput
- VideoInput
- LTCInput
- VITCInput

*logicalUnitNumber:* The number of the input whose name is desired.

*name:* A pointer to the buffer that will receive the input name.

*maxLength:* The length of the buffer.

**Return Value:** TRUE if the name is properly returned; otherwise, FALSE.

**Comments:** The names of the resources can be changed by a system administrator using the Vdr Configuration utility. Unless changed, the default names will be given.

**See Also:** *CfgGetNumInputs* on page 74,
*CfgGetOutputName* on page 80.

## CfgGetInputUnitNum

**#include "tekcfg.h"**

**INT CfgGetInputUnitNum(ConnectHandle** *connHandle*,
**PciInputType** *inputType*,
**CONST CHAR*** *name*)

This function returns the input number based on the input name.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*inputType:* This must be one of the following:

- AudioInput
- VideoInput
- LTCInput
- VITCInput

*name:* The name of an input.

**Return Value:** This function returns the input number.

**See Also**: *CfgGetInputUnitNum* on page 69,
*CfgGetOutputUnitNum* on page 81.

## CfgGetMajorVersion

**#include "tekcfg.h"**

**UINT CfgGetMajorVersion**(**ConnectHandle** *connHandle*)

This function returns the major version number of the TekCfg library. For example, the major number in version 1.23 is **1**.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the major number of the TekCfg library.

## CfgGetMinorVersion

**#include "tekcfg.h"**

**UINT CfgGetMinorVersion(ConnectHandle** *connHandle*)

This function returns the minor version number of the TekCfg library. For example, the minor number in version 1.23 is **23**.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the minor number of the TekCfg library.

## CfgGetNumCodecs

**#include "tekcfg.h"**

**INT CfgGetNumCodecs(ConnectHandle** *connHandle*,
**PciCodecType** *codecType*)

This function returns the total number of codecs of the specified type that are available to all applicaitons.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*codecType:* The possible values are:

- AudioCodec
- JpegCodec (same as VideoCodec)
- TimecodeCodec
- MixEffects
- MpegEncoder
- MpegDecoder
- MpegHdDecoder
- MpegHdEncoder
- Dvcpro25Codec
- Dvcpro50Codec

**Return Value:** This function returns the number of codecs, based on the number and type of boards found at startup and based on software parameters.

**Comments:** Refer to the data sheet for each model to determine the number of audio channels, JPEG codecs, MPEG encoders and decoders, MPEG HD encoders and decoders, and DVCPRO 25 and DVCPRO 50 codecs. These numbers vary between models.

A timecode recorder is capable of recording timecode into a media file, or using a media file to playback previously recorded timecodes. It is a software codec for timecodes. When TimecodeCodec is specified, this function returns the number of timecode recorders allowed.

Mix effects codecs are optional. The currently valid numbers of mix effects codecs are 0, 2, or 4.

Codecs are generic, and are not given names. The application should make up names reflecting the codecType, with a suffix number reflecting the possible codec numbers, 0 through n-1, where the number of codecs is n, for example, VideoCodec0.

**See Also:** *CfgGetCodecAttributes* on page 64, *CfgGetNumInputs* on page 74, and *CfgGetNumOutputs* on page 75.

## CfgGetNumFileSystems

**#include "tekcfg.h"**

**INT CfgGetNumFileSystems**(**ConnectHandle** *connHandle*)

Returns the number of file systems on the embedded system.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the number of file systems into which the media disk space has been partitioned.

**See Also:** *CfgGetFileSystemName* on page 66.

## CfgGetNumInputs

**#include "tekcfg.h"**

**INT CfgGetNumInputs**(**ConnectHandle** *connHandle*,
**PciInputType** *inputType*)

This function returns the number of inputs of the desired type in the PDR system.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*inputType:* This value must be one of the following:

- AudioInput
- VideoInput
- LTCInput
- VITCInput

**Return Value:** This function returns the number of inputs of the type specified, based on the number and type of boards found at startup.

**Comments:** There are different numbers of inputs and/or outputs on different audio and video boards. This function returns the total number of audio or video inputs. The audio and video inputs are numbered consecutively starting at the lowest slot number containing an audio or video board respectively.

There is Longitudinal Timecode capability all systems. Each video input is capable of producing a VITC input.

**See Also:** *CfgGetNumCodecs* on page 72,
*CfgGetNumOutputs* on page 75.

## CfgGetNumOutputs

**#include "tekcfg.h"**

**INT CfgGetNumOutputs**(**ConnectHandle** *connHandle*,
**PciOutputType** *outputType*)

This function returns the count of outputs of the specified type in the PDR system.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*outputType:* This value must be one of the following:

- AudioOutput
- VideoOutput
- LTCOutput
- VITCOutput

**Return Value:** This function returns the number of outputs of the specified type, based on the number and type of boards found at startup.

**Comments:** There are different numbers of inputs and/or outputs on different audio and video boards. This function returns the total number of audio or video outputs. The video outputs are numbered consecutively starting at the lowest slot number containing a video board.

In general, there is a Longitudinal Timecode capability on the Reference GenLock board, and each video output is capable of accepting a VITC output.

**See Also:** *CfgGetNumCodecs* on page 72,
*CfgGetNumInputs* on page 74.

## CfgGetNumResources

**#include "tekcfg.h"**

**Comments:** New command; engineering specification not yet available.

## CfgGetNumTCGenerators

**#include "tekcfg.h"**

**INT CfgGetNumTCGenerators(ConnectHandle** *connHandle*)

This function returns the number of timecode generators in the PDR system.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the number of timecode generators found at startup.

**Comments:** Timecode generators are instances of a software timecode module. The maximum number is determined at the time the software is built. The generator can create timecode values and user bits values in synchronization with the port clock.

## CfgGetNumTCRecorders

**#include "tekcfg.h"**

**INT CfgGetNumTCRecorders**(**ConnectHandle** *connHandle*)

This function returns the number of timecode recorders in the PDR system.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the number of timecode recorders found at startup.

**Comments:** Timecode recorders are instances of a software timecode module. The maximum number is determined at the time the software is built. The recorder can record timecode values and user bits values in synchronization with the port clock.

## CfgGetOutputAttributes

**#include "tekcfg.h"**

**BOOL CfgGetOutputAttributes**(**ConnectHandle** *connHandle*,
**PciOutputType** *outputType*,
**INT** *logicalUnitNumber*,
**PciAttributeTypes_t** *attribute*,
**BOOL*** *result*)

This function returns the attribute of the specified output.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*inputType:* This value must be `AudioOutput`. Additional values will be supported in a future release.

*logicalUnitNumber:* The number of an audio output.

*attribute:* The possible values are:

- LevelAdjustment
- EqualizationAdjustment

*result:* TRUE if the output has the *attribute*; otherwise, FALSE.

**Return Value:** TRUE if successful; otherwise, FALSE.

## CfgGetOutputName

**#include "tekcfg.h"**

**BOOL CfgGetOutputName**(**ConnectHandle** *connHandle*,
**PciOutputType** *outputType*,
**INT** *logicalUnitNumber*,
**CHAR *** *name*,
**INT** *maxLength*)

This function returns the name of the requested output.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*outputType:* The possible values are:

- AudioOutput
- LTCOutput
- VideoOutput
- VITCOutput

*logicalUnitNumber:* The number of the output whose name is desired.

*name:* A pointer to the buffer that will receive the output name.

*maxLength:* The length of the buffer so that the library will not write outside the buffer space.

**Return Value:** TRUE if the name is properly returned; otherwise, FALSE.

**Comments:** The names of the resources can be changed by a system administrator. Unless changed, the default names will be given.

**See Also:** *CfgGetInputName* on page 68,
*CfgGetNumOutputs* on page 75.

## CfgGetOutputUnitNum

**#include "tekcfg.h"**

**INT CfgGetOutputUnitNum**(**ConnectHandle** *connHandle*,
**PciOutputType** *outputType*,
**CONST CHAR*** *name*)

This function returns the output number based on the output name.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*inputType:* The possible values are:

- AudioOutput
- LTCOutput
- VideoOutput
- VITCOutput

*name:* The name of an output.

**Return Value:** This function returns the output number.

**See Also:** *CfgGetInputUnitNum* on page 69,
*CfgGetOutputName* on page 80.

## CfgGetResourceName

**#include "tekcfg.h"**

**Comments:** New command; engineering specification not yet available.

## CfgGetStandard

**#include "tekcfg.h"**

**VideoStandard_t CfgGetStandard(ConnectHandle** *connHandle*)

This function returns the video standard for a Profile system.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** The function will return one of the following:

- PCI_NTSC_525_MODE
- PCI_PAL_625_MODE
- PCI_720_59_94Hz_1To1_MODE
- PCI_720_60Hz_1To1_MODE
- PCI_1080_24Hz_1To1_MODE
- PCI_1080_25Hz_2To1_MODE
- PCI_1080_29_97Hz_2To1_MODE
- PCI_1080_30Hz_2To1_MODE
- PCI_INVALID_MODE

**See Also:** *CfgGetVideoInputStandard* on page 84.

## CfgGetVideoInputStandard

**#include "tekcfg.h"**

**VideoStandard_t CfgGetVideoInputStandard**(**ConnectHandle** *connHandle*, *INT logicalUnitNumber*)

This function returns the video standard for a video input.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.
*logicalUnitNumber:*The number of the video input.

**Return Value:** The function will return one of the following:

- PCI_NTSC_525_MODE
- PCI_PAL_625_MODE
- PCI_720_59_94Hz_1To1_MODE
- PCI_720_60Hz_1To1_MODE
- PCI_1080_24Hz_1To1_MODE
- PCI_1080_25Hz_2To1_MODE
- PCI_1080_29_97Hz_2To1_MODE
- PCI_1080_30Hz_2To1_MODE
- PCI_INVALID_MODE

**See Also:** *CfgGetStandard* on page 83.

# The Pdr library

The TekPdr Library provides an API for enumerating and managing an inventory of Common Movie Format movies and for editing the descriptions of these movies.

## PdrBackupMovieDescription

**#include "tekpdr.h"**

**BOOL PdrBackupMovieDescription**(**ConnectHandle** *connHandle,*
**const CHAR*** *complexMovieName,*
**UINT** *flags*)

This creates a file with the name *complexMovieName.cmf* in the media file system. The CMF description for the named movie is then placed in the file.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*complexMovieName:* The name of the movie to backup.

*flags* Value which controls the backup process. Possible values are:

- PDR_REPLACE_EXISTING;
- PDR_UPDATE_EXISTING;
- PDR_REPLACE_EXISTING | PDR_MAKE_MEDIA_LOCAL; and
- PDR_UPDATE_EXISTING | PDR_MAKE_MEDIA_LOCAL.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** PDR_REPLACE_EXISTING causes an existing *.cmf* file for the movie to be replaced unconditionally; this is usually not necessary.
PDR_UPDATE_EXISTING replaces an existing *.cmf* file only if the Last Change Time is more recent than the Modification Time for the *.cmf* file or if there is no *.cmf* file. If PDR_MAKE_MEDIA_LOCAL is used with either of these flags, any media used by the movie but not on the same dataset will be copied onto that dataset and the *.cmf* description will be changed to use the new media file pathnames.

## PdrCloseAllEnumTokens

**#include "tekpdr.h"**

**BOOL PdrCloseAllEnumTokens**(**ConnectHandle** *hConnect*)

This function closes all enumeration tokens; it is equivalent to multiple calls to the function *PdrCloseFind* but does not require that the actual values of the EnumTokens be known.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** TRUE if the enumeration tokens are all closed; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** This function should be used with care, as it will disrupt the operation of all current enumeration lists.

## PdrCloseFind

**#include "tekpdr.h"**

**BOOL PdrCloseFind(EnumToken** *enumToken*)

This function is used to release the enumeration token used to track the current position within an enumeration list.

**Parameters:**

*enumToken:* The token created with a call to one of the **FindFirst** functions:

- *PdrFindFirstDataset*
- *PdrFindFirstGroup*
- *PdrFindFirstMediaUsage*
- *PdrFindFirstMovie*

**Return Value:** TRUE if the enumeration token is closed; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** There are a limited number of enumeration tokens, and this function lets the library know when a token can be reassigned.

## PdrCloseMovie

**#include "tekpdr.h"**

**BOOL PdrCloseMovie(MovieToken** *movie*)

This function invalidates the use of the movie token for referencing the movie with which it is associated.

**Parameters:**

*movie:* The token returned by a call to *PdrCreateMovie*, *PdrOpenMovie*, or *PdrCopyMovieToken* which created the association.

**Return Value:** TRUE if this reference to the movie is closed; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** When an application terminates, all open movies are automatically closed by the library. When the system first starts, all movies are set to be closed by the library.

## PdrCopyMediaToken

**#include "tekpdr.h"**

**MediaToken PdrCopyMediaToken**(**MediaToken** *existingMediaSegment*,
**MovieToken** *destinationMovie*,
**TrackToken** *destinationTrack*,
**MediaToken** *beforeMedia*)

This function creates a new reference to a media file that has originally been referenced elsewhere. This function provides a way to edit a new movie by including pieces from existing movies. The new movie is the destination; both the movie and the track need to be specified.

**Parameters:**

*existingMediaSegment:* The token of the media that has been obtained by enumeration, or other actions on an open movie.

*destinationMovie:* The token of the movie whose track is being referenced.

*destinationTrack:* A TrackToken for the track being referenced.

*beforeMedia:* The reference media before which the new media will be placed.

**Return Value:** This function returns the MediaToken of the new reference to the media file if there are no errors. Otherwise, the function returns a NULL MediaToken, which can be detected with the function *PdrNullMediaToken*. (For extended error information, use **GetLastError**.)

**Comments:** The actual media file referenced by the *existingMediaSegment* parameter will be shared by the existing **MediaToken** and the new **MediaToken** that is created by this function. This sharing of media files is the most efficient use of the system resources.

If the *beforeMedia* parameter is a NULL MediaToken, the new MediaToken is appended to the end of the track.

If the new MediaToken is inserted onto the track and not appended to the end, the position referenced by the MediaTokens that follow the new MediaToken are shifted to make room for the media referenced by the new MediaToken.

This function creates multiple references to a media file. The references can be from different movies, or can be from the same movie. The library functions ensure that a media file is not deleted while any movie has a reference to that file.

**See Also:** *PdrDeleteMedia* on page 99,
*PdrDeleteMovie* on page 100, and
*PdrNullMediaToken* on page 169.

## PdrCopyMovie

**#include "tekpdr.h"**

**WaitToken PdrCopyMovie(ConnectHandle** *source*,
**const CHAR*** *srcMovie*,
**ConnectHandle** *destination*,
**const CHAR*** *destMovieName*,
**PdrCopy** *copyType*)

This function creates a copy of an existing movie, including the media files. The contents of the existing referenced movie are copied to the newly created movie.

**Parameters:**

*source and destination:* The values returned from *RemOpenConnection*.

*srcMovie:* The complex movie name of the movie which is to be copied.

*destMovieName:* The complex movie name of the destination movie for copy operation.

*copyType:* The variable which controls the copy operation. It may have the values `PdrRenderedMedia` or `PdrExactMedia`. In the case of `PdrRenderedMedia`, only those fields of the media files that are on the timeline of the movie (between the in-point and out-point for the media segment reference) are copied into the new movie's media file. This is more efficient than `PdrExactMedia` which copies all the media files of the source movie.

**Return Value:** If the function succeeds, the return value is the token for the copy operation that will proceed asynchronously. Otherwise, the value zero is returned. (For extended error information, use **GetLastError**.)

**Comments:** Because copying the media files requires a substantial portion of the media file system bandwidth, this function should be used with care. The bandwidth requirement may restrict the ability to play and/or record on all channels simultaneously. Use *PdrCopyMovieToken* which does not copy the actual media files.

This function may also be time-consuming. (The first call may take 300ms or longer, so be careful of using a timeout on the first call.) For this reason, the function is started as an asynchronous task and will not be complete when the function returns. In order to determine the status of the operation and to terminate the operation, the application must call *PdrGetWaitOpStatus* and *PdrTerminateWaitOperation*. The WaitToken returned from this function is used in these related functions to indicate which asynchronous operation is being addressed.

A zero return value can indicate that the reference movie does not exist (the source movie name is bad), or that the destination movie name already exists, or that there are too few resources to start the asynchronous operation.

**See Also:** *PdrCopyMovieToken* on page 91,
*PdrCreateMovie* on page 96,
*PdrDeleteMovie* on page 100,
*PdrGetWaitOpStatus* on page 165, and
*PdrTerminateWaitOperation* on page 193.

## PdrCopyMovieToken

**#include "tekpdr.h"**

**MovieToken PdrCopyMovieToken**(**MovieToken** *existingMovie*,
**const CHAR*** *complexMovieName*)

This function creates a copy of a movie that has been opened and has the *existingMovie* token, and names the new copy the *complexMovieName* given by the input string.

**Parameters:**

*existingMovie:* The movie which is to be copied; notice that it does not need to be in the same group as the new copy.

*complexMovieName*:The name of the new copy of the movie.

**Return Value:** If the function succeeds, the return value is the token for the new movie. Otherwise, the value zero is returned. (For extended error information, use **GetLastError**.)

**Comments:** The new movie will share the media files of the existing movie; each movie will point to the same media files. This is the most efficient manner in which to copy a movie.

The new movie will have its attribute set to not control the ReadOnly attributes and will be ReadWrite if no other movie has made the media ReadOnly.

## PdrCreateBlackMediaToken

**#include "tekpdr.h"**

**MediaToken PdrCreateBlackMediaToken**(**MovieToken** *movie*,
**TrackToken** *track*,
**INT** *length*,
**MediaToken** *beforeMedia*)

This function returns a MediaToken which is a place holder in the media stream and plays BLACK or silence. No media file is actually required. The new MediaToken can be used in other functions.

**Parameters:**

*movie:* The token of the movie that will gain the new media token.

*track:* The TrackToken for the track to which the new MediaToken will be added.

*length:* The initial length of the black media stream in fields; the length can be changed by changing the in- and out-points.

*beforeMedia:* The MediaToken of a reference media before which the new media will be inserted.

**Return Value:** This function returns a MediaToken which can be used in other functions. It returns a special value if either the MovieToken or the TrackToken is invalid. The special value can be determined with the function *PdrMediaTokenIsNull*. (For extended error information, use **GetLastError**.)

**Comments:** The function creates an empty media reference, and places the reference to Black media at the requested point in the movie.

If *beforeMedia* is a Null MediaToken, the new MediaToken is appended to the end of the track, after all other MediaTokens. If the new MediaToken is not the last MediaToken on the track, the media position referenced by the MediaTokens that follow the new MediaToken are shifted to make room for the new media file. The MediaToken in and out values are set to `0` and `length` respectively.

**See Also:** *PdrMediaTokenIsNull* on page 167.

## PdrCreateEvent

**#include "tekpdr.h"**

**Handle PdrCreateEvent**(**ConnectHandle** *connHandle*,
**PdrEventType** *nEventType*)

The function creates a system event that will be set when the selected movie change type occurs.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*nEventType:* One of the types of PDR movies change that will be bound to the created event. Valid types are:

- PdrMovieAllChanges
- PdrMovieAdded
- PdrMovieDeleted
- PdrMovieModified
- PdrMovieGroupsChange

**Return Value:** This function returns a HANDLE to the system event. This handle can be used with WaitForSingleObject, WaitForMultipleObjects, and so forth. Use the CloseHandle function to close the handle. If the operation fails, use **GetLastError** to get extended error information.

**See Also:** *PdrGetMovieChanges* on page 137.

## PdrCreateMediaToken

**#include "tekpdr.h"**

**MediaToken PdrCreateMediaToken**(**MovieToken** *movie*,
**TrackToken** *track*,
**INT** *length*,
**MediaToken** *beforeMedia*)

This function creates a structure to represent a media segment for the movie that has been opened with the *movie* token. The media segment is placed on the track of the movie identified by the *track* token. The media segment will be set so that the in-point is 0 and the out-point is *length*. The actual position of the media segment on the track depends on the value of the *beforeMedia* token.

**Parameters:**

| | |
|---|---|
| *movie:* | The token of the movie that will gain the new media token. |
| *track:* | The TrackToken for the track to which the new MediaToken will be added. |
| *length:* | The initial length of the new media file in fields; the length can be changed by changing the in-point and out-point. |
| *beforeMedia:* | The MediaToken of a reference media before which the new media will be inserted. |

**Return Value:** This function returns a MediaToken which can be used in other functions. It returns a special value if either the MovieToken or the TrackToken is invalid. The special value can be determined with the function *PdrMediaTokenIsNull*. (For extended error information, use **GetLastError**.)

**Comments:** If the *beforeMedia* parameter is a NULL MediaToken, the new MediaToken is appended to the end of the track, after all other MediaTokens. If the *beforeMedia* token is a valid MediaToken for an existing media segment on the referenced track, the new media segment is placed in front of the referenced *beforeMedia* segment. If the *beforeMedia* token does not reference an existing media segment on the reference track, an error occurs and the function cannot be completed.

If the new MediaToken is inserted onto the track and not appended to the end, the position referenced by the MediaTokens that follow the new MediaToken are shifted to make room for the media referenced by the new MediaToken. The MediaToken in and out values are set to `0` and `length` respectively.

The new media segment references a media file. The file will not exist until it is attached to a Resource allocated to a Port. (See TekVdr library functions.) At that time, the file is added to the file system. The file pathname will reflect the movie's group and name, the track on which the media segment is created, and the number of media segments already created on this track. The file will be created on the file system reflected in the movie's dataset value. (For example, a media segment created for the movie with dataset INT;, group Profile, and name Test1, and on Audio track 1 would have a file pathname of the general form: INT:/Profile/Test1-003.A1. The order in which media segments are created is independent of their order on the track.)

**See Also:** *PdrMediaTokenIsNull* on page 167.

## PdrCreateMediaTokenWithPath

**#include "tekpdr.h"**

**MediaToken PdrCreateMediaToken(MovieToken** *movie,*
**TrackToken** *track,*
**INT** *length,*
**MediaToken** *beforeMedia,*
**const CHAR*** *mediaPath*)

The function creates a structure to represent a media segment for the movie that has been opened with the *movieToken*. The media segment is placed on the track of the movie identified by the *track* Token. The media segment will be set so that its in-point is 0, and its out-point is *length*. The actual position of the media segment on the track depends on the value of the *beforeMedia* token. The media is in the file specified by *mediaPath*; that file must be a Vfs file of the correct type. If the file does not exist yet, it will be created when the movie/media is attached to the Port Timeline.

**Parameters:**

| | |
|---|---|
| *movie:* | The token of the movie that will gain the new media token. |
| *track:* | The TrackToken for the track to which the new MediaToken will be added. |
| *length:* | The initial length of the new media file in fields; the length can be changed by changing the in-point and out-point. |
| *beforeMedia:* | The MediaToken of a reference media before which the new media will be inserted. |
| *mediaPath*: | The full pathname of the media file to be used for this media segment. |

**Return Value:** If the function cannot be completed, a special MediaToken of value Null is returned. The Null value Media Token can be detected using the return value from the function *PdrMediaTokenIsNull*. It returns a Boolean TRUE or FALSE after testing the MediaToken for the special value representation. (For extended error information, use **GetLastError**.)

**Comments:** If the *beforeMedia* token is the special representation of Null, the new media segment will be appended to the end of any other media segments on the referenced track. If the *beforeMedia* token is a valid MediaToken for an existing media segment on the reference track, the new media segment is placed in front of the referenced *beforeMedia* segment. If the existing media segment is not on the reference track, an error occurs and the function cannot be completed.

The new media segment represents the media file with the name given by the *mediaPath*. If the file does not already exist, it will be created when it is attached to the Port resource for Record or Play. At that time, the file is added to the file system.

Because the filename is created outside the library, it may be impossible for the utilities that validate the correctness of the common movie format with the files in the Video File System to correlate the file to the media segment independently.

**See Also:** *PdrMediaTokenIsNull* on page 167.

## PdrCreateMovie

**#include "tekpdr.h"**

**MovieToken PdrCreateMovie**(**ConnectHandle** *connHandle*,
**const CHAR *** *complexMovieName*)

This function creates the named movie and returns a token that can be used to reference the new movie.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*complexMovieName:* The name string for the new movie.

**Return Value:** If the function succeeds, the return value is the token for the named movie. Otherwise, the value zero is returned. (For extended error information, use **GetLastError**.)

**Comments:** A return value of `0` can indicate that the specific movie name already exists.

The new movie contains neither resource references, nor media.

The new movie will be opened exclusive for the creator.

**See Also:** *PdrCopyMovie* on page 90,
*PdrDeleteMovie* on page 100.

## PdrCreateTrackToken

**#include "tekpdr.h"**

**TrackToken PdrCreateTrackToken**(**ConnectHandle** *connHandle*,
**PdrTrackType** *trackType*,
**INT** *logicalNumber*)

This function creates a TrackToken for a specific track. The TrackToken refers to tracks within a movie.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*trackType:* The type of track for which the ResourceToken is desired. The possible PdrTrackType values are:

- PdrJpegVideoTrack (same as PdrVideoTrack)
- PdrMpegVideoTrack
- PdrMpegHiDefVideoTrack
- PdrDvcpro25Track
- PdrDvcpro50Track
- PdrAudioTrack
- PdrTimecodeTrack

*logicalNumber:* The number of the specific track of the type to be referenced. The number of tracks of each type is available from the function **PdrGetMaxTracks**.

**Return Value:** This function returns a TrackToken value. If the function fails, it returns `0`. (For extended error information, use **GetLastError**.)

**Comments:** A TrackToken does not depend upon any specific movie. The TrackToken is an independent representation of the specific track.

## PdrDeleteExtensionAtPos

**#include "tekpdr.h"**

**BOOL PdrDeleteExtensionAtPos**( **const PdrExtensionPos*** *pos* );
This function removes the extension from the movie structure.

**Parameters:**

*pos*: The enumeration structure returned by the call to **PdrFindFirstExtensionPos** or **PdrFindNextExtensionPos**.

**Return Value:** The function returns TRUE if the extension is successfully removed. Otherwise the function returns FALSE. To get extended error information use the **GetLastError** function.

## PdrDeleteMedia

**#include "tekpdr.h"**

**BOOL PdrDeleteMedia**(**MediaToken** *mediaSegment*)

This function deletes the reference to a media segment within a movie and may delete the media file containing the media segment.

**Parameters:**

*mediaSegment:* The reference to a segment of the media.

**Return Value:** TRUE if the referenced media token is deleted; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** MediaTokens are unique for each segment in each movie and exist only within the process address space in which they are generated. They are either created or obtained through the enumeration of the media segments in an open movie.

The function deletes a MediaToken from the movie and track that are implicitly bound to that MediaToken. Each MediaToken is an indirect reference to a media file; if the count of references to the media file goes to zero, the media file is also deleted.

If the deleted MediaToken is not the last (latest in time) MediaToken for a track, the positions of the other media tokens are shifted so as to compact the track and to eliminate any holes. The shift is always from later times to earlier times in the movie.

**See Also:** *PdrDeleteMovie* on page 100.

## PdrDeleteMovie

**#include "tekpdr.h"**

**BOOL PdrDeleteMovie**(**ConnectHandle** *connHandle*,
**const CHAR*** *complexMovieName*)

This function deletes a movie.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.
*complexMovieName:* The name of the movie to be deleted.

**Return Value:** TRUE if the function succeeds; otherwise, FALSE. The function fails if the *complexMovieName* does not reference a movie. (For extended error information, use **GetLastError**.)

**Comments:** Once deleted, the movie can no longer be referenced or used. Deleting the movie can cause the media files to be deleted also. If the files are not referenced by any other movie, they are deleted. A movie must be closed before it can be deleted. Movies in use by others can not be deleted.

A return value of FALSE may also indicate an inconsistency between the common movie format description and the files in the media file system.

## PdrFindFirstDataset

**#include "tekpdr.h"**

**EnumToken PdrFindFirstDataset**(**ConnectHandle** *connHandle*,
**CHAR*** *rtnBuffer*,
**UINT** *bufferSize*)

This function initiates the enumeration of the known datasets. The function places the name of the first dataset in the buffer supplied by the caller, and returns an EnumToken that can be used to get other dataset names.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *rtnBuffer:* | The buffer used to receive the dataset name. |
| *bufferSize*: | The size of the buffer. |

**Return Value:** A valid EnumToken is greater than zero. An EnumToken with any other value indicates that an error occurred while trying to access the enumeration list. (For extended error information, use **GetLastError**.)

**Comments:** There are a limited number of EnumTokens maintained by the library. If the library does not have an available EnumToken to assign this enumeration list, an error is returned.

**See Also:** *PdrCloseFind* on page 87,
*PdrFindNextDataset* on page 106.

## PdrFindFirstExtensionPos

**#include "tekpdr.h"**

**BOOL PdrFindFirstExtensionPos( int** *firstField*,
**int** *lastField*,
**const PdrExtensionInfo*** *pSearchInfo*,
**UINT*** *dataLen*,
**PdrExtensionPos*** *pos* );

This function initiates the enumeration of extensions in a particular movie. The enumeration starts by choosing which extension attributes to use as search criteria. Only extensions that match this criteria will appear in the enumeration. Search criteria may include a specific location in the movie structure indicated by *movie*, *track*, and *media* values in the PdrExtensionInfo structure; may be limited to a specific range of fields relative to the specified location; or may be comprised of one or more key values pointed to by *pSearchInfo* including *vendorName*, *dataClass*, *dataType*, *dataName*, and *dataVersion*. The **PdrExtensionPos** structure filled-in by this function contains the search criteria as well as an index that will allow you to extract the actual extension data with a subsequent call to **PdrGetExtensionAtPos**.

**Parameters:**

*firstField*: Extensions which fall outside the range set by *firstField* and *lastField* will be excluded from this enumeration. The field values are relative to the movie location (movie/track/mediaSegment). A value of –1 indicates all fields before the *lastField* are valid in the search.

*lastField*: Defines the end of the search range. A value of –1 indicates all fields after the firstField are valid to the search.

*pSearchInfo*: Pointer to **PdrExtensionInfo** structure. All fields in this structure must be filled in prior to the call to **PdrInsertExtension**. Empty strings in this structure are interpreted as “don't care.”

*dataLen*: Number of bytes in the extension data structure at the current position, zero (0) if an extension could not be found.

*pos*: This structure contains all the search criteria entered by this function and is used for subsequent calls to **PdrFindNextExtensionPos** and **PdrGetExtensionAtPos**.

**Return Value:** The function returns TRUE if at least one extension that matches the search criteria could be found. Otherwise the function returns FALSE. To get extended error information use the **GetLastError** function.

**Comments:** If you want to enumerate all extensions associated with a particular *mediaSegment*, fill in values for movie, track, and *mediaSegment*. Set *firstField* and *lastField* to –1 and pSearchInfo's *vendorName*, *dataClass*, *dataType*, *dataName*, and *dataVersion* to "".

If you wish to enumerate all extensions in the movie, set track to 0 and *mediaSegment* to **PdrNullMediaToken**, *firstField* and *lastField* to –1 and all other values in *pSearchInfo* to "".

When the **PdrExtensionPos** structure is initialized, the movie token value is copied. Do not close the movie until you are done making calls to extension functions that use this structure.

## PdrFindFirstGroup

**#include "tekpdr.h"**

**EnumToken PdrFindFirstGroup**(**ConnectHandle** *connHandle*,
**const CHAR*** *startPoint*,
**CHAR*** *rtnBuffer*,
**UINT** *bufferSize*)

This function initiates the enumeration of the groups within a dataset, starting with a group specified in the *startPoint* string. The *function places the name of a group in the buffer supplied by the caller, and returns an EnumToken that can be used to get other group names.*

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *startPoint*: | A complex name string without a movie name part; it may contain a dataset part and/or a group part. |
| *rtnBuffer:* | The buffer used to receive the group name. |
| *bufferSize*: | The size of the return buffer. |

**Return Value:** A valid EnumToken greater than zero. An EnumToken with any other value indicates that an error occurred while trying to access the enumeration list. (For extended error information, use **GetLastError**.)

The function will return a `0` value EnumToken if there are no groups in the dataset, or if the search for the starting group string passes the end of the dataset's groups.

**Comments:** If the dataset part is given, it must be a valid dataset. The given group part must be a full group name. The first item in the list that is equal to or greater than the starting group string will be returned. If the dataset part is *not* given, the value of the current dataset is used.

There are a limited number of EnumTokens maintained by the library. If the library does not have an available EnumToken to assign this enumeration list, an error is returned.

**See Also:** *PdrCloseFind* on page 87,
*PdrFindNextGroup* on page 108.

## PdrFindFirstMediaUsage

**#include "tekpdr.h"**

**EnumToken PdrFindFirstMediaUsage (ConnectHandle** *hConnect,*
**const char*** *szComplexMediaName,*
**PdrMediaUsage*** *pMediaUsage,*
**UINT** *pdr_db_version)*

This function initiates the enumeration of a list of movie names that contain references to the given media file name. The function places the name of the movie in the buffer supplied by the caller, and returns an EnumToken that can be used to get other movie names.

**Parameters:**

*hConnect:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*szComplexMediaName*:
The full path to a media file in the form [datasetdir]/PDR/[groupdir]/[filename].

*pMediaUsage:* A structure that receives the complex media names. The structure is defined as:

```
typedef struct
{
    INT         in;
    INT         out;
    CHARdset[PDR_MAX_DSET_NAME_LEN+1];
    CHARgroup[PDR_MAX_GROUP_NAME_LEN+1];
    CHARname[PDR_MAX_MOVIE_NAME_LEN+1];
}
PdrMediaUsage;
```

The in value represents the first field of the given media file that is played by this movie. The out value represents the field after the last field that is played.

*pdr_db_version*: The Pdr database version number.

**Return Value:**

**Comments:** A complex media name is the full path to a media file: [datasetdir]/PDR/[groupdir]/[filename], for example, EXT:/PDR/default/clip12.M0. The term is the media equivalent to the "complex movie name." Complex media names are returned by calls such as PdrGetMediaPath and PdrGetMediaStateInfo.

Close the enumeration token using *PdrCloseFind* when you are done.

**See Also:** *PdrCloseFind* on page 87,
*PdrFindNextMediaUsage* on page 109.

## PdrFindFirstMovie

**#include "tekpdr.h"**

**EnumToken PdrFindFirstMovie(ConnectHandle** *connHandle*,
**const CHAR*** *startPoint*,
**CHAR*** *rtnBuffer*,
**UINT** *bufferSize*)

This function initiates the enumeration of the movies within a group, starting with a movie specified in the *startPoint* string. It places the name of a movie in the buffer supplied by the caller, and returns an EnumToken that can be used to get other movie names.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*startPoint*: A complex name string. It may contain a dataset part, a group part, and/or a movie part.

*rtnBuffer:* The buffer used to receive the first movie name.

*bufferSize*: The size of the buffer.

**Return Value:** A valid EnumToken is greater than zero. An EnumToken with any other value indicates that an error occurred while trying to access the enumeration list. (For extended error information, use **GetLastError**.)

The function will return a `0` value EnumToken if there are no movies in the group, or if the search for the starting movie name string passes the end of the group.

**Comments:** If the dataset part is given, it must be a valid dataset. The given group part must be a valid group. The movie part should be a full movie name. The first item in the list equal to or greater than the starting group string will be returned. If the dataset part is *not* given, the value of the current dataset is used; if the group part is not given, the value of the current group is used.

There are a limited number of EnumTokens maintained by the library. If the library does not have an available EnumToken to assign this enumeration list, an error is returned.

**See Also:** *PdrCloseFind* on page 87,
*PdrFindNextGroup* on page 108.

## PdrFindNextDataset

**#include "tekpdr.h"**

**BOOL PdrFindNextDataset**(**EnumToken** *enumToken*,
**CHAR*** *rtnBuffer*,
**UINT** *bufferSize*)

This function continues the enumeration of the datasets started with the *PdrFindFirstDataset* function. The enumeration returns the next dataset in the list to which the EnumToken belongs.

**Parameters:**

*enumToken:* The token returned by a call to the *PdrFindFirstDataset*.

*rtnBuffer:* The buffer used to receive the next dataset name.

*bufferSize*: The size of the buffer.

**Return Value:** If there are more datasets in this enumeration list, the function places the name of the next dataset in the buffer supplied by the caller, and returns the boolean value TRUE. A FALSE return value indicates either that the last dataset name has been returned previously, or that an error occurred while trying to access the next item in the enumeration list. (For extended error information, use **GetLastError**.)

**See Also:** *PdrCloseFind* on page 87,
*PdrFindFirstDataset* on page 101.

## PdrFindNextExtensionPos

**#include "tekpdr.h"**

**BOOL PdrFindNextExtensionPos**( **UINT*** *dataLen*,
**PdrExtensionPos*** *pos* );

This function continues the enumeration that was started with the **PdrFindFirstExtensionPos** function. The enumeration returns the next extension that matches the search criteria contained in the structure pointed to by *pos*.

**Parameters:**

*dataLen*: Number of bytes in the extension data structure at the current position. Zero if an extension could not be found.

*pos*: This structure contains the search criteria entered by the original call to **PdrFindFirstExtensionPos** and is used for subsequent calls to **PdrFindNextExtensionPos** and **PdrGetExtensionAtPos**.

**Return Value:** The function returns TRUE if at least one extension that matches the search criteria could be found. Otherwise the function returns FALSE. To get extended error information use the **GetLastError** function.

## PdrFindNextGroup

**#include "tekpdr.h"**

**BOOL PdrFindNextGroup(EnumToken** *enumToken*,
**CHAR*** *rtnBuffer*,
**UINT** *bufferSize*)

This function continues the enumeration of the groups started with the *PdrFindFirstGroup* function. The enumeration returns the next group in the list to which the EnumToken belongs.

**Parameters:**

*enumToken:* The token returned by a call to the *PdrFindFirstGroup*.

*rtnBuffer:* The buffer used to receive the next group name.

*bufferSize*: The size of the buffer.

**Return Value:** If there are more groups in this enumeration list, the function places the name of the next group in the buffer supplied by the caller, and returns the boolean value TRUE. A FALSE return value indicates either that the last group name has been returned previously, or that an error occurred while trying to access the next item in the enumeration list. (For extended error information, use **GetLastError**.)

**See Also:** *PdrCloseFind* on page 87,
*PdrFindFirstGroup* on page 103.

## PdrFindNextMediaUsage

**#include "tekpdr.h"**

**BOOL PdrFindNextMediaUsage (EnumToken** *enumToken,*
**PdrMediaUsage*** *pMediaUsage,*
**UINT** *pdr_db_version***)**

This function continues the enumeration of media usage started with the *PdrFindFirstMediaUsage* function. The enumeration returns the next media usage in the list to which the EnumToken belongs.

**Parameters:**

*enumToken:* The token returned by a call to the *PdrFindFirstMediaUsage*.

*pMediaUsage:* A structure that receives the complex media names. The structure is defined as:

```
typedef struct
{
    INT         in;
    INT         out;
    CHARdset[PDR_MAX_DSET_NAME_LEN+1];
    CHARgroup[PDR_MAX_GROUP_NAME_LEN+1];
    CHARname[PDR_MAX_MOVIE_NAME_LEN+1];
}
PdrMediaUsage;
```

The in value represents the first field of the given media file that is played by this movie. The out value represents the field after the last field that is played.

*pdr_db_version*: The Pdr database version number.

**Return Value:**

**Comments:**

**See Also:** *PdrCloseFind* on page 87,
*PdrFindFirstMediaUsage* on page 104.

## PdrFindNextMovie

**#include "tekpdr.h"**

**BOOL PdrFindNextMovie(EnumToken** *enumToken*,
**CHAR*** *rtnBuffer*,
**UINT** *bufferSize*)

This function continues the enumeration of the movies started with the *PdrFindFirstMovie* function. The enumeration returns the next movie in the list to which the EnumToken belongs.

**Parameters:**

*enumToken:* The token returned by a call to the *PdrFindFirstMovie*.

*rtnBuffer:* The buffer used to receive the next movie name.

*bufferSize*: The size of the buffer.

**Return Value:** If there are more movies in this enumeration list, the function places the name of the next movie in the buffer supplied by the caller, and returns the boolean value TRUE. A FALSE return value indicates either that the last movie name has been returned previously, or that an error occurred while trying to access the next item in the enumeration list. (For extended error information, use **GetLastError**.)

**See Also:** *PdrCloseFind* on page 87,
*PdrFindFirstMovie* on page 105.

## PdrFreeExtension

**#include "tekpdr.h"**

**BOOL PdrFreeExtension**( **MovieToken** *movie*,
**VOID*** *dataPtr* );

This function frees the memory allocated with **PdrReadExtension**.

**Parameters:**

*movie*: The token for the movie with which the extension was registered.

*dataPtr*: Pointer to the buffer returned by **PdrReadExtension**.

**Return Value:** The function returns TRUE if the memory is successfully freed. Otherwise the function returns FALSE. To get extended error information use the **GetLastError** function.

## PdrGetCurrentDataset

**#include "tekpdr.h"**

**BOOL PdrGetCurrentDataset**(**ConnectHandle** *connHandle*,
**CHAR*** *rtnBuffer*,
**UINT** *bufferSize*)

This function acquires the string that is being used as the current dataset. This string is used in several functions to convert complex movie names in which the dataset part is missing.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *rtnBuffer:* | The buffer used to receive the current dataset. |
| *bufferSize*: | The size of the buffer. |

**Return Value:** TRUE if no errors are detected; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The current dataset value is set using the *PdrSetCurrentDataset* function. The default value of the current dataset is the first file system of the Profile; this value is used if the function *PdrSetCurrentDataset* is never called.

## PdrGetCurrentGroup

**#include "tekpdr.h"**

**BOOL PdrGetCurrentGroup**(**ConnectHandle** *connHandle*,
**CHAR*** *rtnBuffer*,
**UINT** *bufferSize*)

This function acquires the string that is being used as the current group. This string is used in several functions to convert complex movie names in which the group part is missing.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *rtnBuffer:* | The buffer used to get the current group. |
| *bufferSize*: | The size of the buffer. |

**Return Value:** TRUE if no errors are detected; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The current group value is set using the *PdrSetCurrentGroup* function. The default value of the current group is “default”; this value is used if the function *PdrSetCurrentGroup* is never called.

## PdrGetExtensionAtPos

**#include "tekpdr.h"**

**BOOL PdrGetExtensionAtPos( const PdrExtensionPos*** *pos*,
**PdrExtensionInfo*** *pInfo*,
**UINT** *dataLen*,
**void*** *dataPtr*);

This functions returns the extension data for the given extension position.

**Parameters:**

*pos*: The enumeration structure returned by the call to **PdrFindFirstExtensionPos** or **PdrFindNextExtensionPos**.

*pInfo*: The buffer used to receive the **PdrExtensionInfo** structure.

*dataLen*: Number of bytes in the extension data buffer. If this value is less than the actual size of the extension, then only the first 'dataLen' bytes of the extension will be copied.

*dataPtr*: The buffer used to receive the extension data structure.

**Return Value:** The function returns TRUE if the extension data can be successfully returned. Otherwise the function returns FALSE. To get extended error information use the **GetLastError** function.

## PdrGetExtensionInfoAtPos

**#include "tekpdr.h"**

**BOOL PdrGetExtensionInfoAtPos( const PdrExtensionPos*** *pos,*
**PdrExtensionInfo*** *pInfo* );

This functions returns the header information for the extension at the given position.

**Parameters:**

*pos*: The enumeration structure returned by the call to **PdrFindFirstExtensionPos** or **PdrFindNextExtensionPos**.

*pInfo*: The buffer used to receive the **PdrExtensionInfo** structure.

**Return Value:** The function returns TRUE if the extension header can be successfully returned. Otherwise the function returns FALSE. To get extended error information use the **GetLastError** function.

## PdrGetFirstTcPosition

**#include "tekpdr.h"**

**INT PdrGetFirstTcPosition**(**ConnectHandle** *connHandle*,
**const CHAR*** *complexMovieName*,
**UINT** *nTcTrackNum*,
**UINT** *nTimeCode*)

This function returns the earliest position in a timecode media file that contains the specified timecode.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*complexMovieName:* This value specifies the movie to be examined.

*nTcTrackNum*: This value specifies the number of the timecode track to be examined.

*nTimeCode*: This value specifies the timecode value for matching. See *VdrGetCurrentTimeCode* for a description of timecode values.

**Return Value:** This function returns the field position in the media file containing the earliest occurrence of the timecode value or `-1,` which indicates an error. (For extended error information, use **GetLastError**.)

**Comments:** The movie must be a simple clip with a single media segment for the timecode. The timecode recorded in the media file should be continuous and either repeating or increasing.

The timecode track cannot have been created by “striping”.

## PdrGetFullName

**#include "tekpdr.h"**

**BOOL PdrGetFullName(ConnectHandle** *connHandle*,
**const CHAR** **name*,
**CHAR** **pFname*,
**INT** *size*)

This will convert the PDR movie name (**const CHAR** **name*) to its native name on the server. The full native name is returned in the buffer *pFname* if the call succeeds.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| **name:* | The PDR movie name. |
| **pFname:* | A buffer to contain the movie name in the server native format. |
| *size:* | The size of the buffer. |

**Return Value:** TRUE if it successfully obtains a movie name; otherwise, FALSE. Check GetLastError to see why it failed.

The full movie name returned by the server is a directory. The directory has the following files present:

| | |
|---|---|
| attr | movie attribute file |
| index | index file |
| stream | the movie as a byte stream |
| done | zero length file indicating the movie is complete. If this file does not exist, the stream is not complete. |

If a third party is restoring a stream, the files should be restored in the order shown above.

**Comments:** This call is *only* supported by File Server connected to a Lightspeed Server; if used with a Profile, *this call will fail*.

## PdrGetLastTcPosition

**#include "tekpdr.h"**

**INT PdrGetLastTcPosition**(**ConnectHandle** *connHandle,*
**const CHAR*** *complexMovieName,*
**UINT** *nTcTrackNum,*
**UINT** *nTimeCode*)

This function returns the last position in a timecode media file that contains the specified timecode.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*complexMovieName:* Specifies the movie to be examined.

*nTcTrackNum*: Specifies the logical number of the timecode track to be examined.

*nTimeCode*: Specifies the timecode value for matching. See *VdrGetCurrentTimeCode* for a description of timecode values.

**Return Value:** This function returns the field position in the media file containing the last occurrence of the timecode value or `-1` which indicates an error. (For extended error information, use **GetLastError**.)

**Comments:** The movie must be a simple clip with a single media segment for the timecode. The timecode recorded in the media file should be continuous, and either repeating or increasing.

The timecode track cannot have been created by “striping”.

## PdrGetListOfNames

**#include "tekpdr.h"**

**INT PdrGetListOfNames**(**ConnectHandle** *connHandle*,
**const CHAR*** *szStart*,
**INT** *nStartIndex*,
**CHAR*** *szRtnBuffer*,
**INT** *nLength*)

This returns a list of names for existing datasets, groups, or movies, depending on the supplied input parameters.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*szStart:* A case-sensitive string used to designate the Pdr object to use when forming the list of names. Given a dataset name, all groups of the dataset would be listed; given a group name bounded by forward slashes (/), all movies of the group would be listed; given Null, all datasets would be listed.

*nStartIndex*: A zero-based index of where to start listing names.

*szRtnBuffer*: Pointer to a memory buffer in which to place as many names as will fit. The names placed in this buffer are Null terminated strings placed one after another, with the final name being double Null terminated.

*nLength*: The length, in bytes, of the parameter *szRtnBuffer*.

**Return Value:** This function returns the number of names that were copied into the return buffer. A return value of zero is returned if no names were found. A negative return value indicates an error occurred. (For extended error information, use **GetLastError**.)

**Comments:** The return buffer must be large enough to hold one name.

**See Also:** *PdrCloseFind* on page 87, *PdrFindFirstDataset* on page 101, *PdrFindFirstGroup* on page 103, *PdrFindFirstMovie* on page 105, *PdrFindNextDataset* on page 106, *PdrFindNextGroup* on page 108, and *PdrFindNextMovie* on page 110.

**Example 1. Print all groups in all datasets**

```
PrintAllGroups()
{
   CHAR buffer[80];
   CHAR dataset[PDR_MAX_DSET_NAME_LEN + 1];
   EnumToken enumToken= PdrFindFirstDataset(NULL, dataset, PDR_MAX_DSET_NAME_LEN);
   if (enumToken <= 0) {
      return;
}
do {
      UINT nTotal= 0; // Start at name index 0
      INT nFound;
      while ((nFound= PdrGetListOfNames(LOCAL_CONNECTION, dataset, nTotal, buffer,
                80)) > 0) {
```

```
        if (nFound < 0)
              break;
        nTotal += nFound; // Start next iteration here
        CHAR* ptr= buffer;
        while (nFound--) {
              printf("%s/%s/\n", dataset, ptr);
              INT len= strlen(ptr) + 1;
              ptr += len; // Point to next name
         }
     }
}
while (PdrFindNextDataset(enumToken, dataset, PDR_MAX_DSET_NAME_LEN));
PdrCloseFind(enumToken);
}
```

## PdrGetMajorVersion

**#include "tekpdr.h"**

**UINT PdrGetMajorVersion(ConnectHandle** *connHandle*)

This returns the integral value of the library MajorVersion.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the major part of the version number of the PdrMovie library.

**Comments:** The value of the library version is the same value as is shown by the file properties as “version” to the left of the decimal point. The version number is used to identify changes in the library implementation.

## PdrGetMaxTracks

**#include "tekpdr.h"**

**UINT PdrGetMaxTracks**(**ConnectHandle** *connHandle*,
**PdrTrackType** *trackType*)

Returns the number of tracks of a specific type supported by the system. Any logical track number for a track of this type must have a value that is less than the maximum number of tracks of the type.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*trackType:* The type of track for which the maximum number is desired. The possible PdrTrackType values are:

- PdrJpegVideoTrack (same as PdrVideoTrack)
- PdrMpegVideoTrack
- PdrMpegHiDefVideoTrack
- PdrDvcpro25Track
- PdrDvcpro50Track
- PdrAudioTrack
- PdrTimecodeTrack

**Return Value:** This function returns the maximum number of tracks supported by the system. If the operation fails, the function returns a value of `-1`. (For extended error information, use **GetLastError**.)

## PdrGetMediaAttributes

**#include "tekpdr.h"**

**PdrAttributes PdrGetMediaAttributes**(**MediaToken** *media*)

This function returns the attributes of the media segment.

**Parameters:**

*media:* The indirect reference to the media file being queried.

**Return Value:** This function returns the attribute value of the media file being queried. If an error occurs, the value **PdrError** is returned. Use the function **GetLastError** to obtain extended error information.

**Comments:** The attributes are represented by a bit-mask; valid attributes for the media segment are **PdrReadOnly** and **PdrControlRO**. The **PdrControlRO** bit is set if the movie containing this media segment was explicitly set to ReadOnly. This bit indicates that the media files ReadOnly attributes should not be changed to ReadWrite unless this movie is changed to ReadWrite access.

The ReadOnly bit reflects the libraries knowledge of the access control on the media file; it is less expensive to recover than the actual file system status. However, if the access control is changed using lower level functions, the libraries media attributes will not reflect the files access control.

## PdrGetMediaDataset

**#include "tekpdr.h"**

**INT PdrGetMediaDataset**(**MediaToken** *media*,
**CHAR*** *rtnBuffer*,
**UINT** *bufferSize*)

This function returns the path of the referenced media file. If the dataset string that is returned does not compare equal to the file system name(s) of the local file system(s), the media will be located remotely in the network of systems. It might need to be copied to the local system before being played or recorded.

**Parameters:**

| | |
|---|---|
| *media:* | The token for the media file whose dataset name is desired. |
| *rtnBuffer:* | A pointer to the buffer into which the dataset name is to be stored. |
| *bufferSize:* | The size of the buffer into which the dataset name is to be stored. |

**Return Value:** This function returns the number of characters in the dataset name. If an error occurs, the value `-1` is returned. (For extended error information, use **GetLastError**.)

**Comments:** A return value of `-1` can indicate that MediaToken is invalid.

## PdrGetMediaDataType

**#include "tekpdr.h"**

**FileType PdrGetMediaDataType**(**MediaToken** *media*)

This function returns the type of media that is recorded in the referenced file. The possible data types are:

- FileTypeJpegVideo (same as FileTypeVideo)
- FileTypeMpegVideo
- FileTypeMpeHiDefVideo
- FileTypeDvcpro25Video
- FileTypeDvcpro50Video
- FileTypeAudio
- FileTypeTimecode
- FileTypeNonMedia

**Parameters:**

*media:* The token for the media file whose data type is desired.

**Return Value:** This function returns the type of media. If an error occurs, the function returns the FileType code FileTypeNonMedia. (For extended error information, use **GetLastError**.)

## PdrGetMediaDuration

**#include "tekpdr.h"**

**INT PdrGetMediaDuration**(**MediaToken** *mediaToken*)

This function returns the number of field positions that will be occupied by the media segment referenced by the media token. The media duration is specified as the field value of OUT minus the field value of IN; the duration is the number of fields of media that will be played or recorded.

**Parameters:**

*media:* The token for the media file whose duration is desired.

**Return Value:** This function returns the duration that will be occupied by the media segment referenced by the media token. If an error occurs, the value `-1` is returned. (For extended error information, use **GetLastError**.)

## PdrGetMediaIn

**#include "tekpdr.h"**

**INT PdrGetMediaIn**(**MediaToken** *media*)

This function returns the IN value for the specified media token.

**Parameters:**

*media:* The token for the media file whose IN value is desired.

**Return Value:** This function returns the IN value for the specified media token. If an error occurs, the value `-1` is returned. (For extended error information, use **GetLastError**.)

**Comments:** The IN value is the first field in the file that will be used when the media file is on the Port Timeline. The field in the media file with the value IN will be the first field recorded and played. Media tokens that are of the type PdrMediaBLACK also have IN values; normally the IN value is set to zero since there is no file being referenced.

Field numbers in media files start at 0. The IN value of a media token must be positive, and must be less than or equal to the OUT value. If the IN value is equal to the OUT value, no fields from the media file will participate in record/playback activity.

## PdrGetMediaMarks

**#include "tekpdr.h"**

**BOOL PdrGetMediaMarks**(**MediaToken** *media,*
**INT*** *inFieldNum,*
**INT*** *outFieldNum*)

This function returns the in-point and out-point for the specified media token.

**Parameters:**

| | |
|---|---|
| *media:* | The token for the media file whose in-point and out-point are desired. |
| *inFieldNum:* | The variable into which the in-point is to be stored. |
| *outFieldNum:* | The variable into which the out-point is to be stored. |

**Return Value:** TRUE if there are no errors; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The fields in a media file are numbered from 0. Media files are addressable at the field level.

The in-point of a media file is the first field that is recorded or played. The out-point of a media file is the first field after the last field to be recorded or played.

## PdrGetMediaOut

**#include "tekpdr.h"**

**INT PdrGetMediaOut(MediaToken** *media*)

This function returns the OUT value for the specified media token.

**Parameters:**

*media:* The token for the media file whose OUT value is desired.

**Return Value:** This function returns the OUT value for the specified media token. If an error occurs, the value `-1` is returned. (For extended error information, use **GetLastError**.)

**Comments:** The OUT value is the field number plus one of the last field in the file that will be used when the media file is on the Port Timeline. The last field that is played or recorded is `OUT -1`. Media tokens that are of the type PdrMediaBLACK also have OUT values; normally the OUT value is the length in fields for which black media will be played. This assumes that these media tokens have their IN values set to `0`.

Field numbers in media files start at 0. The out-point value must be positive and it must be greater than or equal to the in-point value. If the out-point value is equal to the in-point value, no fields from the media file will participate in record or playback activity when the movie is attached to a Port Timeline.

## PdrGetMediaPath

**#include "tekpdr.h"**

**INT PdrGetMediaPath**(**MediaToken** *mediaToken*,
**CHAR*** *rtnBuffer*,
**UINT** *bufferSize*)

This function fills in the pathname to the media file reference by the specified media token.

**Parameters:**

*mediaToken:* The token for the media file whose pathname is desired.
*rtnBuffer:* A pointer to the buffer into which the pathname is to be stored.
*bufferSize:* The size of the buffer into which the pathname is to be stored.

**Return Value:** This function returns the number of characters in the file pathname. If an error occurs, the value `-1` is returned. (For extended error information, use **GetLastError**.)

**Comments:** Only media tokens of type PdrMediaFile reference media files. The full pathname of a media file is expected to be less than PDR_MAX_COMPLEX_MEDIA_NAME_LEN characters.

## PdrGetMediaStateInfo

**#include "tekpdr.h"**

**BOOL PdrGetMediaStateInfo**(**MovieToken** *movie,*
**TrackToken** *track,*
**MediaToken** *referenceMedia,*
**PdrWhichMedia** *whichMedia,*
**PdrMediaState*** *rtnStruct,*
**UINT** *databaseVersion*)

This function returns all of the available information about a media segment. This single function eliminates the need to make multiple queries using conventional function calls.

**Parameters:**

*movie:* The attached MovieToken for the queried media state.

*track:* The attached TrackToken for the queried track state.

*referenceMedia:* A MediaToken value that describes the media segment that is having its state queried.

*whichMedia:* An enumeration value that indicates the media relative to the *referenceMedia* value that is having its state queried. The enumeration values are:

| | |
|---|---|
| -1 | PdrPrevMedia |
| 0 | PdrThisMedia |
| 1 | PdrNextMedia |

*rtnStruct:* A pointer to a **PdrMediaState** structure into which the state information can be stored.

*databaseVersion:* The value, defined in the *pdrversion.h* header file, which identifies the version of the database. The version number is changed whenever the database structure is changed.

**Return Value:** If the function succeeds, it fills in the PdrMediaInfo structure and returns the Boolean value TRUE. If the function fails, the values in the PdrMediaInfo structure are undefined, and it returns the Boolean value FALSE. (For extended error information, use **GetLastError**.)

**Comments:** This function, supported on all platforms for which the TekPdr library is supported, allows retrieval of the media path and IN/OUT values in a single function call, reducing overhead. It also returns the values of adjacent MediaTokens to minimize the overhead of looping through all media segments on a track.

**PdrGetMediaStateInfo** is the same as the earlier **PdrGetMediaState**, with the addition of the *databaseVersion* parameter. The obsolete **PdrGetMediaState** command can still be called; this would be equivalent to calling **PdrGetMediaStateInfo** with a *databaseVersion* parameter of `PDR_DB_VERSION`.

**See Also:** *PdrGetMediaAttributes* on page 123,
*PdrGetMediaIn* on page 127,
*PdrGetMediaMarks* on page 128,

*PdrGetMediaOut* on page 129,
*PdrGetMediaPath* on page 130,
*PdrGetMediaTokenType* on page 134,
*PdrGetNextMediaToken* on page 149,
*PdrGetPositionOfMediaToken* on page 154
*PdrGetPreviousMediaToken* on page 155.

## PdrGetMediaTokenAtPosition

**#include "tekpdr.h"**

**MediaToken PdrGetMediaTokenAtPosition**(**MovieToken** *movie*,
**TrackToken** *track*,
**INT** *fieldNum*)

This function gets the MediaToken of the media to be active at a specific position.

**Parameters:**

| | |
|---|---|
| *movie:* | The token of the movie containing the media of interest. |
| *track:* | The TrackToken of the track containing the media of interest. |
| *fieldNum:* | The position on the movie timeline for which the MediaToken is desired. |

**Return Value:** This function returns a media token that can be used to reference the media file or to identify the in-point and out-point in a file. It returns a Null MediaToken that can be detected with the *PdrMediaTokenIsNull* function if either the MovieToken, or the TrackToken is invalid, or if the position is beyond the scope of any media file on the specified track. (For extended error information, use **GetLastError**.)

**Comments:** A position specifies the relative time at which the media file referenced by a MediaToken would playback if the movie were played from its beginning. Each track contains one or more MediaTokens that describe media files, and how they are to be used. The position value of a MediaToken correlates the field numbers of the movie's clock with the field numbers of the media files referenced by the MediaToken. The position value of a MediaToken defines the movie clock time at which the media file's field marked by the in value would be played. The clock starts with field number 0 at the beginning of the movie.

**See Also:** *PdrGetPositionOfMediaToken* on page 154.

## PdrGetMediaTokenType

**#include "tekpdr.h"**

**PdrMediaTokenType PdrGetMediaTokenType**(**MediaToken** *media*)

This function returns the **PdrMediaTokenType** of the media token.

**Parameters:**

*media:* The token whose type is desired.

**Return Value:** The type of the media token is returned; on an error, **PdrError** is returned. (For extended error information, use **GetLastError**.)

**Comments:** Media Tokens are of the type PdrMediaFile or PdrMediaBlack. Only tokens of type PdrMediaFile reference a media file and contain a media path.

## PdrGetMinorVersion

**#include "tekvdr.h"**

**UINT PdrGetMinorVersion(ConnectHandle** *connHandle*)

This function returns the fraction value of the library version.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the minor part of the version number of the PdrMovie library.

**Comments:** The fraction value of the library version is the same value as is shown by the file properties as “version” to the right of the decimal point. The version number is used to identify changes in the library implementation.

## PdrGetMovieAttributes

**#include "tekpdr.h"**

**PdrAttributes PdrGetMovieAttributes**(**MovieToken** *movie*)

This function returns a mask containing bits which can be tested to determine the attributes of a specified movie.

**Parameters:**

*movie:* The token for the movie whose attributes are desired.

**Return Value:** If the function succeeds, the return value is the attributes of the movie. Otherwise, the value PdrError is returned. (For extended error information, use **GetLastError**.)

**Comments:** A PdrError return value can indicate that *movieToken* is invalid.

The following are attributes for a movie:

| | |
|---|---|
| PdrAudio16Bit | PdrSampleRate30 |
| PdrAudio24Bit | PdrSampleRate50 |
| PdrCodecConstruction | PdrSampleRate60 |
| PdrControlRO | PdrSimpleClip |
| PdrCopyConstruction | PdrStreamConstruction |
| PdrExtended | PdrTcDropFrame |
| PdrLocked | PdrTcNonDropFrame |
| PdrOpen | PdrUnderConstruction |
| PdrOpenExclusive | PdrVideoFormatMPEG* |
| PdrOpenMultiple | PdrVideoFormatMpegHiDef |
| PdrReadOnly | PdrVideoFormatDVCPRO25 |
| PdrRestoreConstruction | PdrVideoFormatDVCPRO50 |
| PdrSampleRate24 | PdrVideoFormatJPEG |
| PdrSampleRate25 | PdrVideoFormatMpegStdDef |

*PdrVideoFormatMPEG is supported in version 4.2 and higher for backward compatibility. PdrVideoFormatMpegStdDef is preferred.

**See Also:** *pdrattrib.h.* for mask values and their meanings.

## PdrGetMovieChanges

**#include "tekpdr.h"**

**INT PdrGetMovieChanges(ConnectHandle** *connHandle*,
**PdrEventType** *nEventType*,
**PdrEventInfo*** *pStruct*,
**INT** *nCount*)

This function returns information from the buffer of change notifications.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*nEventType:* The type of PDR movie changes to be copied into the space provided by the parameter *pStruct*. Valid types are:

- PdrMovieAdded
- PdrMovieAllChanges
- PdrMovieDeleted
- PdrMovieGroupChanged
- PdrMovieInvalidEvent
- PdrMovieModified

*nCount:* The number of structure members that *pStruct* contains.

*pStruct:* An array of structures that will receive the movie change information. The structure is defined as:

```
typedef struct
{
   PdrEventType nEventType:
   FILETIME eventTime;
   ULONG nModifyBitmask;
   CHAR szMovieName[PDR_MAX_COMPLEX_MOVIE_NAME_LEN + 1];
}
PdrEventInfo;
```

The nModifyBitmask can contain the following values:

For nEventType **PdrMovieAdded**:

- PDR_NOTIFY_MOVIE_ADDED

For nEventType **PdrMovieDeleted**:

- PDR_NOTIFY_MOVIE_DELETED

For nEventType **PdrMovieGroupChanged**:

- PDR_NOTIFY_GROUP_ADDED
- PDR_NOTIFY_GROUP_DELETED

For nEventType **PdrMovieModified**:

- PDR_NOTIFY_CONTROL_RO
- PDR_NOTIFY_CONSTRUCTION
- PDR_NOTIFY_LENGTH
- PDR_NOTIFY_LOCKED
- PDR_NOTIFY_MARKIN
- PDR_NOTIFY_MARKOUT
- PDR_NOTIFY_MEDIA_ADDED
- PDR_NOTIFY_MEDIA_DELETED
- PDR_NOTIFY_RENAME_NEW_NAME
- PDR_NOTIFY_RENAME_OLD_NAME
- PDR_NOTIFY_READ_ONLY
- PDR_NOTIFY_READ_WRITE
- PDR_NOTIFY_THUMBNAIL
- PDR_NOTIFY_USERDATA

**Return Value:** This function returns the number of PdrEventInfo structures that contain valid information. A return of `0` indicates there is no information available. A negative number indicates there was an error. **GetLastError** may return `PDR_SOME_NOTIFICATION_DATA_DISCARDED` if some movie change information was lost due to buffer overflow.

**Comments:** A rename will always be composed of a movie notification on `OLD_NAME` followed by a movie notification on `NEW_NAME`, without any intervening notifications. One notification will never occur without the other.

**See Also:** *PdrCreateEvent* on page 93.

**Example 2. Print movie deletions**

```
PrintMovieDeletes()
{
   HANDLE hEvent= PdrCreateEvent(NULL, PdrMovieDeleted);

   while (WaitForSingleObject(hEvent, INFINITE) != WAIT_OBJECT_0) {
      INT nChangesReturned;
      PdrEventInfo eventInfo[3];
      while (nChangesReturned= PdrGetMovieChanges(NULL, PdrMovieDeleted, eventInfo,
                  3)) {
      for (INT i= nChangesReturned; i; --i) {
         printf("Deleted Movie: %s\n", eventInfo[i].szMovieName);
      }
      }
   }
}
```

## PdrGetMovieCreateTime

**#include "tekpdr.h"**

**BOOL PdrGetMovieCreateTime**(**MovieToken** *movie*,
**FILETIME ***_time_)

This function returns the time a specified movie was created.

**Parameters:**

| | |
|---|---|
| *movie:* | The token for the movie whose create time is wanted. |
| *time:* | The location of the data structure that is to receive the *time* value. |

**Return Value:** TRUE if the function succeeds; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** A FALSE return value can indicate that *movieToken* is invalid.

**See Also:** Win32 FILETIME.

## PdrGetMovieDataset

**#include "tekpdr.h"**

**INT PdrGetMovieDataset**(**MovieToken** *movieToken*,
**CHAR ***rtnBuffer*,
**UINT** *bufferSize*)

This function fills in the buffer with the name of the dataset used by the movie.

**Parameters:**

| | |
|---|---|
| *movieToken:* | The token for the movie whose dataset name is desire. |
| *rtnBuffer:* | A pointer to the buffer in which the dataset name is to be stored. |
| *bufferSize:* | The size of the buffer into which the dataset name is to be copied. |

**Return Value:** This function returns the number of characters in the dataset name. If an error occurs, the value `0` is returned. (For extended error information, use **GetLastError**.)

**Comments:** The dataset name will be one of the FileSystem names returned by calls to the function *CfgGetFileSystemName*. The default dataset is FileSystem 0.

## PdrGetMovieGroup

**#include "tekpdr.h"**

**INT PdrGetMovieGroup**(**MovieToken** *movie*,
**CHAR *** *buffer*,
**UINT** *bufferLength*)

This function returns the group name of the specified movie.

**Parameters:**

| | |
|---|---|
| *movieToken:* | The token for the movie whose group name is desired. |
| *buffer:* | A pointer to the buffer into which the group name is to be stored. |
| *bufferLength:* | The length of the buffer into which the group name is to be copied. |

**Return Value:** If the function succeeds, the return is the actual length of the group name. Otherwise, the value `0` is returned. (For extended error information, use **GetLastError**.)

**Comments:** Movie groups are created as needed and deleted when no longer needed.

The default group is “default”.

## PdrGetMovieLastChangeTime

**#include "tekpdr.h"**

**BOOL PdrGetMovieLastChangeTime**(**MovieToken** *movie,*
**FILETIME *** *time*)

This function returns the FILETIME of the last change to the movie data.

**Parameters:**

*movie:* The token for the movie whose last modification time is desired.

*time:* A pointer to the location into which the FILETIME of the last change should be stored.

**Return Value:** TRUE if the function succeeds; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The last change time is updated whenever the movie structure is changed in a manner that might effect the application. For example, when the attributes change and when the length changes, the last change time is updated.

A FALSE return value can indicate that *movie* is invalid.

**See Also:** Win32 FILETIME.

## PdrGetMovieLength

**#include "tekpdr.h"**

**INT PdrGetMovieLength**(**MovieToken** *movie,*
**PdrLengthType** *lengthType*)

This function returns the length (in fields) of the specified movie using the mode desired.

**Parameters:**

*movieToken:* The token for the movie whose length is desired.

*lengthMode:* The type of length to be returned; this can be either PdrMinLength, or PdrMaxLength.

**Return Value:** If the function succeeds, it returns the length of the movie in fields, which could be zero. A return value of `-1` can indicate an error. (For extended error information, use **GetLastError**.)

**Comments:** A `-1` return value can indicate that *movie* is invalid. Movies with no media attached will have a length of 0.

A movie will normally have more than a single media track and the tracks will not be identical in length. This function returns either the maximum length of all the tracks (PdrMaxLength), or the minimum track length (PdrMinLength).

## PdrGetMovieMarkIn

**#include "tekpdr.h"**

**INT PdrGetMovieMarkIn**(**MovieToken** *movie*)

This function returns the current movie level mark IN value for the desired movie.

**Parameters:**

*movie:* The token for the movie for which an IN mark is desired.

**Return Value:** If an error occurs, the value `-1` is returned. (For extended error information, use **GetLastError**.)

**Comments:** The movie level mark IN is available for a named movie in the PdrMovieState structure. A movie does not need to be open to get its movie marks.

The movie marks are initially set during “Record” by the update thread in TekVdr. The movie marks are used to “Attach” the movie to the timeline.

## PdrGetMovieMarkOut

**#include "tekpdr.h"**

**INT PdrGetMovieMarkOut**(**MovieToken** *movie*)

This function returns the current movie level mark OUT value for the desired movie.

**Parameters:**

*movie:* The token for the movie for which an OUT mark is desired.

**Return Value:** If an error occurs, the value `-1` is returned. (For extended error information, use **GetLastError**.)

**Comments:** The movie level mark OUT is available for a named movie in the PdrMovieState structure. A movie does not need to be open to get its movie marks.

The movie marks are initially set during “Record” by the update thread in TekVdr. The movie marks are used to “Attach” the movie to the timeline.

## PdrGetMovieName

**#include "tekpdr.h"**

**INT PdrGetMovieName**(**MovieToken** *movie*,
**CHAR ***buffer*,
**UINT** *bufferSize*)

This function returns the name of the specified movie.

**Parameters:**

| | |
|---|---|
| *movie:* | The token for the movie whose name is desired. |
| *buffer:* | A pointer to the buffer into which the name is to be stored. |
| *bufferSize:* | The size of the buffer into which the name is to be copied. |

**Return Value:** If the function succeeds, the return is the actual length of the name (the number of characters in the name). Otherwise, the value `-1` is returned. (For extended error information, use **GetLastError**.)

**Comments:** A `-1` return value can indicate that *movie* is invalid.

## PdrGetMovieStateInfo

**#include "tekpdr.h"**

**BOOL PdrGetMovieStateInfo(ConnectHandle** *connHandle*,
**const CHAR*** *complexMovieName*,
**PdrMovieState*** *pStateStruct*,
**UINT** *databaseVersion*)

This function provides information about a stored movie without requiring that the movie be open.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*complexMovieName:* The name of the movie whose state is desired.

*pStateStruct:* A pointer to a **PdrMovieState** structure into which the movie state information is stored.

*databaseVersion:* The value, defined in the *pdrversion.h* header file, which identifies the version of the database. The version number is changed whenever the database structure is changed.

**Return Value:** TRUE if the movie exists and the state can be stored in the state structure; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** Use of this function can reduce the number of open movies an application might require since simple browsing will not require that the movie be open in order to acquire its parameterization.

**PdrGetMovieStateInfo** is the same as the earlier **PdrGetMovieState**, with the addition of the *databaseVersion* parameter. The obsolete **PdrGetMovieState** command can still be called; this would be equivalent to calling **PdrGetMovieStateInfo** with a *databaseVersion* parameter of `PDR_DB_VERSION`.

**See Also:** *PdrGetMovieAttributes* on page 136,
*PdrGetMovieLength* on page 143,
*PdrGetMovieMarkIn* on page 144,
*PdrGetMovieMarkOut* on page 145,
*PdrGetMovieTokenFromMediaToken* on page 148.

## PdrGetMovieTokenFromMediaToken

**#include "tekpdr.h"**

**MovieToken PdrGetMovieTokenFromMediaToken**(**MediaToken** *media*)

This function returns the movie token that is associated with the specific media token.

**Parameters:**

*media:* The MediaToken assigned.

**Return Value:** This function returns the MovieToken for the referenced MediaToken. If the MediaToken is invalid, the function returns 0. (For extended error information, use **GetLastError**.)

**Comments:** A MediaToken contains an implicit binding to a specific movie and track.

## PdrGetNextMediaToken

**#include "tekpdr.h"**

**MediaToken PdrGetNextMediaToken(MovieToken** *movie*,
**TrackToken** *track*,
**MediaToken** *referenceMedia*)

This function returns a MediaToken from the specified track of the specified movie.

**Parameters:**

| | |
|---|---|
| *movieToken:* | The token of the movie for which the MediaToken is desired. |
| *track:* | A TrackToken for the track on which the MediaToken is to be found. |
| *referenceMedia:* | The reference media from which to find the next MediaToken. |

**Return Value:** This function returns a MediaToken which can be used in other functions. The function returns a special value if either the MovieToken or the ResourceToken is invalid, or there is no valid MediaToken to return. The special value can be determined with the function *PdrMediaTokenIsNull*. (For extended error information, use **GetLastError**.)

**Comments:** The function allows the enumeration of the MediaTokens on a track. If the *referenceMedia* parameter is a NULL MediaToken, the function returns the first MediaToken on the track. If the *referenceMedia* parameter is the last MediaToken on the track, the function returns a NULL MediaToken which can be detected with the function *PdrMediaTokenIsNull*.

A NULL MediaToken can be created with *PdrNullMediaToken*.

**See Also:** *PdrGetPreviousMediaToken* on page 155,
*PdrMediaTokenIsNull* on page 167, and
*PdrNullMediaToken* on page 169.

## PdrGetNextTrack

**#include "tekpdr.h"**

**TrackToken PdrGetNextTrack**(**MovieToken** *movie*,
**TrackToken** *referenceTrack*)

This enumerates the tracks of a movie. Each TrackToken returned is used as an input parameter (*referenceTrack*) in the function call to return the next track token.

**Parameters:**

| | |
|---|---|
| *movie:* | The token of the movie for which the track enumeration is desired. |
| *referenceTrack:* | A TrackToken for the track from which the next track is to be referenced. |

**Return Value:** This function returns a TrackToken which can be used in other function calls. The function returns 0 if either the MovieToken or the reference TrackToken is invalid. If the reference track supplied by the *referenceTrack* parameter is the last track of the movie, the return value will also be `0`. (For extended error information, use **GetLastError**.)

**Comments:** If the reference track parameter *referenceTrack* is 0, the function returns the TrackToken of the first track of the specified movie.

**See Also:** *PdrGetPreviousTrack* on page 156.

## PdrGetNumMediaOnTrack

**#include "tekpdr.h"**

**INT PdrGetNumMediaOnTrack**(**MovieToken** *movie*,
**TrackToken** *referenceTrack*)

This function returns the number of media tokens on the specified track. The media tokens can represent either the use of a segment of a media file or the use of BLACK media.

**Parameters:**

*movie:* The token of the movie for which the number of media tokens is desired.

*referenceTrack:* A TrackToken of the track for which the number of media tokens is desired.

**Return Value:** This function returns the number of media tokens on the referenced track. The function returns `-1` if either the MovieToken or the ResourceToken is invalid. (For extended error information, use **GetLastError**.)

**Comments:** Each reference to a different media file, or to a different contiguous section of a media file, requires a MediaToken to specify the information. Movies that contain more than a simple recording will have multiple media tokens on any track to reflect the editing operations.

## PdrGetNumMovies

**#include "tekpdr.h"**

**UINT PdrGetNumMovies**(**ConnectHandle** *connHandle*,
**const CHAR*** *evaluationArea*)

This function returns the number of movies in the area to be evaluated. The evaluation area can be a dataset, or a group within a dataset.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*evaluationArea:* A Complex Movie name string. The movie name part must be empty.

**Return Value:** If the dataset part of the Complex Movie Name is empty, the value of the current dataset is used. If the group part is empty, the evaluation returns the number of movies in all groups of the dataset. If the group part is not empty, the evaluation returns the number of movies in the specified group.

## PdrGetNumTracks

**#include "tekpdr.h"**

**INT PdrGetNumTracks(MovieToken** *movie*,
**PdrTrackType** *trackType*)

This function returns the number of tracks of a specific type used by the movie.

**Parameters:**

*movie:* The token of the movie for which the track count is desired.

*trackType:* The type of track for which the track count is desired. The possible PdrTrackType values are:

- PdrJpegVideoTrack (same as PdrVideoTrack)
- PdrMpegVideoTrack
- PdrMpegHiDefVideoTrack
- PdrDvcpro25Track
- PdrDvcpro50Track
- PdrAudioTrack
- PdrTimecodeTrack

**Return Value:** This function returns the number of tracks of the desired type used by the specified movie. The function returns -1 if either *movie* or *trackType* is invalid. (For extended error information, use **GetLastError**.)

**Comments:** This function may be used to determine if the port resources are adequate to play or record the movie.

There can be as many as 32 tracks associated with a particular track type. However, it is likely the physical resources of the system will be depleted before that large of a number is reached.

## PdrGetPositionOfMediaToken

**#include "tekpdr.h"**

**INT PdrGetPositionOfMediaToken**(**MediaToken** *mediaToken*)

This function gets the Position value of a MediaToken.

**Parameters:**

*media:* The token of the media being referenced.

**Return Value:** This function returns the position value of the referenced media token. It returns -1 if the media token is invalid. (For extended error information, use **GetLastError**.)

**Comments:** The Position value specifies the relative time at which the media file referenced by a MediaToken would playback if the movie were played from its beginning. Each track contains one or more MediaTokens that describe media files, and how they are to be used. The Position value of a MediaToken correlates the field numbers of the movie's clock with the field numbers of the media files referenced by the MediaToken. The Position value of a MediaToken defines the movie's clock time at which the media file's field marked by the in value would be played. The movie's clock starts with field number 0 at the beginning of the movie. (This is not to be confused with the clock of the TekVdr; the movie's position on its timeline is not necessarily at 0, but could be at either a positive or negative TekVdr clock value).

## PdrGetPreviousMediaToken

**#include "tekpdr.h"**

**MediaToken PdrGetPreviousMediaToken(MovieToken** *movieToken,*
**TrackToken** *track,*
**MediaToken** *referenceMedia*)

This function returns a MediaToken from the specified track of the specified movie.

**Parameters:**

*movieToken:* The token of the movie for which the MediaToken is desired.

*track:* A TrackToken for the track on which the MediaToken is to be found.

*referenceMedia:* The reference media from which to find the previous MediaToken.

**Return Value:** This function returns a MediaToken which can be used in other functions. The function returns a special value if either the MovieToken or the TrackToken is invalid, or there is no valid MediaToken to return. The special value can be determined with the function *PdrMediaTokenIsNull*. (For extended error information, use **GetLastError**.)

**Comments:** The function allows the enumeration of the MediaTokens on a track. If the *referenceMedia* parameter is a Null MediaToken, the function returns the last MediaToken on the track. If the *referenceMedia* parameter is the first MediaToken on the track, the function returns a Null MediaToken which can be detected with the function *PdrMediaTokenIsNull*.

A NULL MediaToken can be created with *PdrNullMediaToken*.

**See Also:** *PdrGetNextMediaToken* on page 149,
*PdrMediaTokenIsNull* on page 167, and
*PdrNullMediaToken* on page 169.

## PdrGetPreviousTrack

**#include "tekpdr.h"**

**TrackToken PdrGetPreviousTrack**(**MovieToken** *movie*,
**TrackToken** *referenceTrack*)

This function enumerates the tracks of a stored movie to identify the track tokens. Each TrackToken returned is used as an input parameter (*referenceTrack*) in the function call to return the next track token.

**Parameters:**

*movie:* The token of the movie for which the track enumeration is desired.

*referenceTrack:* The reference track from which the previous track is found.

**Return Value:** This function returns a TrackToken which can be used in other function calls. The function returns 0 if either the MovieToken or referenceTrack value is invalid. If the reference track given by the *referenceTrack* parameter is the first track of the movie, the return value will be `0`. (For extended error information, use **GetLastError**.)

**Comments:** To find the last track of a movie, set the *referenceTrack* token to 0.

**See Also:** *PdrGetNextTrack* on page 150.

## PdrGetRegistry

**#include "tekvdr.h"**

**BOOL PdrGetRegistry**(**ConnectHandle** *connHandle*,
**PdrRegistry*** *regValues*)

This function gets the current PdrMovie configuration settings from the NT Registry.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*regValues:* The address of a PdrRegistry structure that will be used to store the registry values. The structure is defined as:

```
typedef struct
{
   UINT MaxMovies;/* Maximum number of PdrMovies */
   UINT MaxMediaRef;/* Maximum number of Media References */
   UINT MinMediaDef;/* Minimum number of Media Definitions */
}
PdrRegistry;
```

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** When the actual number of movies or media references reaches ten percent of the values specified in the registry, TekPdr will log an NT event warning. When this occurs, an application may either delete unwanted Movies and Media References or increase the maximum values by these parameters by calling the *PdrSetRegistry* function.

The value for Media Definitions provides a default size for the Movie Definition. Media and tracks can be added without concern to this default size.

Some of the movie database is mapped into Vitrual Memory at all times. Very large values for MaxMovies and MaxMediaRef reduce the amount of virtual memroy available for applications and can increase paging and reduce responsiveness.

**See Also:** *PdrSetRegistry* on page 190.

**Example 3. Get the current PdrRegistry settings**

```
GetPdrRegistryValues()
{
   PdrRegistry RegVal;

   if (PdrGetRegistry(NULL, &RegVal))
      printf("Max Movies = %d Max Media Ref = 5d Min Media Def = %d\n",
                  RegVal.MaxMovies, RegVal.MaxMediaRef, RegVal.MinMediaDef);
   else
      printf("Error %d getting PdrRegistry values\n",GetLastError());
}
```

## PdrGetResourceTypeFromTrackType

**#include "tekpdr.h"**

**ResourceType PdrGetResourceTypeFromTrackType**(**ConnectHandle** *connHandle*, **PdrTrackType** *track*)

This function enables you to convert the logical tracks of a movie in the inventory to the resources of the port timeline.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*track:* A **PdrTrackType** value to be converted to a ResourceType value. The possible PdrTrackType values are:

- PdrJpegVideoTrack (same as PdrVideoTrack)
- PdrMpegVideoTrack
- PdrMpegHiDefVideoTrack
- PdrDvcpro25Track
- PdrDvcpro50Track
- PdrAudioTrack
- PdrTimecodeTrack

**Return Value:** This function returns a TekVdr ResourceType value for the given track type (see *Table 20, Vdr resource types* on page 283). If the function fails, it returns the value ResourceInvalid. (For extended error information, use **GetLastError**.)

**Comments:** PdrTrackType values are a subset of all of the ResourceType values. This function provides a translation between the PdrTrackType value and the ResourceType.

**See Also:** *PdrGetTrackTokenType* on page 162.

## PdrGetTrackLength

**#include "tekpdr.h"**

**INT PdrGetTrackLength(MovieToken** *movie*,
**TrackToken** *track*)

This function returns the length of the specified track.

**Parameters:**

| | |
|---|---|
| *movie:* | The token of the movie for which the track length is desired. |
| *track:* | A TrackToken of the track for which the length is desired. |

**Return Value:** This function returns the length of the referenced track. The function returns -1 if either the MovieToken or the TrackToken is invalid. (For extended error information, use **GetLastError**.)

**Comments:** The individual tracks of a movie can be different lengths, reflecting the operations done at the track level. A track without any media attached has a length of 0.

**See Also:** *PdrGetMovieLength* on page 143.

## PdrGetTrackTokenFromMediaToken

**#include "tekpdr.h"**

**TrackToken PdrGetTrackTokenFromMediaToken**(**MediaToken** *mediaToken*)

This function returns the track token associated with the specified media token.

**Parameters:**

*mediaToken:* The token of the media for which the track token is desired.

**Return Value:** This function returns the TrackToken for the referenced MediaToken. If the MediaToken is invalid, the function returns `0`. (For extended error information, use **GetLastError**.)

**Comments:** The media token can be used to identify the movie, the track, and the position in the stored movie.

## PdrGetTrackTokenNum

**#include "tekpdr.h"**

**INT PdrGetTrackTokenNum**(**TrackToken** *trackToken*)

This function returns the logical unit number used to create the track token.

**Parameters:**

*trackToken:* The TrackToken of interest.

**Return Value:** This function returns the logical unit number used to create the track token. The value `-1` is returned if an error is detected. Use the function **GetLastError** to obtain extended error information.

## PdrGetTrackTokenType

**#include "tekpdr.h"**

**PdrTrackType PdrGetTrackTokenType**(**TrackToken** *trackToken*)

This function returns the type of track that was used to create the track token.

**Parameters:**

*trackToken:* A TrackToken for the track of interest.

**Return Value:** Returns a track type. The possible PdrTrackType values are:

- PdrJpegVideoTrack (same as PdrVideoTrack)
- PdrMpegVideoTrack
- PdrMpegHiDefVideoTrack
- PdrDvcpro25Track
- PdrDvcpro50Track
- PdrAudioTrack
- PdrTimecodeTrack

If the function fails, it returns PdrTrackInvalid. (For extended error information, use **GetLastError**.)

## PdrGetTrackTypeFromResourceType

**#include "tekpdr.h"**

**PdrTrackType PdrGetTrackTypeFromResourceType**(**ConnectHandle** *connHandle*, **ResourceType** *resource*)

This function returns the type of track needed by a specific Port Timeline resource.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*resource:* A tekvdr ResourceType value to be converted to a PdrTrackType value.

**Return Value:** Returns a track type. The possible PdrTrackType values are:

- PdrJpegVideoTrack (same as PdrVideoTrack)
- PdrMpegVideoTrack
- PdrMpegHiDefVideoTrack
- PdrDvcpro25Track
- PdrDvcpro50Track
- PdrAudioTrack
- PdrTimecodeTrack

If the ResourceType does not have a corresponding PdrTrackType, a value of `PdrTrackInvalid` is returned. (For extended error information, use **GetLastError**.)

**Comments:** Most resources do not have an equivalent track type in the movie, and the value returned will be `PdrTrackInvalid`. Only resources to which media files can be attached have a corresponding PdrTrackType values in the movie description. This function does the conversion.

## PdrGetUserData

**#include "tekpdr.h"**

**BOOL PdrGetUserData**(**MovieToken** *movie*,
**TrackToken** *track*,
**MediaToken** *mediaSegment*,
**UINT** *idTag*,
**VOID*** *value*,
**UINT** *valueLen*)

This function gets the data bytes that were previously saved using the function *PdrSetUserData*.

**Parameters:**

| | |
|---|---|
| *movie:* | A reference to the movie associated with the user data. |
| *track:* | A reference to the track of a movie associated with the user data. |
| *mediaSegment:* | A reference to the specific media segment associated with the user data. |
| *idTag:* | A unique, programmer-supplied value that differentitates the specific user data from other user data stored with the same movie, track, and media segment. The *idTag* is comprised of two parts: a vendor value and the item value. |
| *value:* | Pointer to the buffer where the user data bytes will be placed. |
| *valueLen:* | The size of the buffer. |

**Return Value:** This function returns TRUE if data as identified by the key composed of *movie*, *track*, *mediaSegment*, and *idTag* exists and was placed in the buffer identified by value. If the function fails, it returns FALSE. (For extended error information, use **GetLastError**.)

**Comments:** To retrieve data, the key must have been set using *PdrSetUserData* and must be identical to the key used when saving the data. If the user data is associated with the movie but not the specific track or media segment, the value track will be `0`. If the data is associated with a specific track but not a specific media segment, the value *PdrNullMediaToken* returns is used as the *mediaSegment* portion of the key.

**See Also:** *PdrSetUserData* on page 192.

## PdrGetWaitOpStatus

**#include "tekpdr.h"**

**BOOL PdrGetWaitOpStatus(WaitToken** *wait*,
**PdrWaitOpStatus*** *pStatus*)

This function gets the status of an asynchronous operation started by *PdrCopyMovie* or *PdrSaveMovie*.

**Parameters:**

*wait:* A WaitToken returned by either *PdrCopyMovie* or *PdrSaveMovie* function. This value must be non-zero.

*pStatus:* A pointer to a *PdrWaitOpStatus* structure that is to be filled by this function.

**Return Value:** TRUE if the function was able to get a status for the operation; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** See *PdrTerminateWaitOperation* for closing the completed operation or for canceling an incomplete operation.

**See Also:** *PdrCopyMovie* on page 90,
*PdrSaveMovie* on page 175, and
*PdrTerminateWaitOperation* on page 193.

## PdrInsertExtension

**#include "tekpdr.h"**

**BOOL PdrInsertExtension( const PdrExtensionInfo*** *pInfo,*
**const void*** *dataPtr* );

This function copies the data pointed to by *dataPtr* into the movie's list of extensions. The extension is tagged by its location and attribute values set in the **PdrExtensionInfo** structure pointed to by *pInfo*.

**Parameters:**

*pInfo*: Pointer to PdrExtensionInfo structure. All fields in this structure must be filled in prior to the call to **PdrInsertExtension**.

*dataPtr*: A pointer to the extension data structure which is copied into the movie structure.

**Return Value:** The function returns TRUE if the extension was correctly inserted. Otherwise the function returns FALSE. To get extended error information use the **GetLastError** function.

**Comments:** The Extension Specification does not require an extension's location or attributes to be unique. It is up to the software that uses extensions to handle duplicates.

## PdrMediaTokenIsNull

**#include "tekpdr.h"**

**BOOL PdrMediaTokenIsNull(MediaToken** *mediaToken*)

This function evaluates a MediaToken to determine if it is the special value representing a Null MediaToken.

**Parameters:**

*mediaToken:* The MediaToken whose value is in question.

**Return Value:** TRUE if the MediaToken has a Null value; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** A MediaToken with the value Null is a special representation of the MediaToken. This function tests for the special representation. An invalid MediaToken will also return a FALSE value.

**See Also:** *PdrNullMediaToken* on page 169.

## PdrMovieExists

**#include "tekpdr.h"**

**BOOL PdrMovieExists**(**ConnectHandle** *connHandle*,
**const CHAR*** *complexMovieName*)

This function determines if the referenced movie has a Common Movie Specification.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*complexMovieName:* The name of the movie in question.

**Return Value:** TRUE if the movie exists; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## PdrNullMediaToken

**#include "tekpdr.h"**

**MediaToken PdrNullMediaToken**(void)

This function returns a MediaToken with a special value.

**Parameters:** None.

**Return Value:** This function returns a MediaToken with a special value used represent Null.

**Comments:** A MediaToken with the value Null is a special representation of the MediaToken. This function generates the special representation.

**See Also:** *PdrMediaTokenIsNull* on page 167.

## PdrOpenMovie

**#include "tekpdr.h"**

**MovieToken PdrOpenMovie**(**ConnectHandle** *connHandle*,
**const CHAR*** *complexMovieName*,
**UINT** *openFlags*)

This function expects a complex movie name string as input.If the structure describing the movie is available and not already exclusively open, it will be opened.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*complexMovieName:* The name of the movie to be opened.

*openFlags:* The tokens indicating how the movie is to be opened.

**Return Value:** If function succeeds, the movie will be opened and a MovieToken representing the movie is returned. In cases where the movie cannot be opened, a MovieToken of value `0` is returned. (For extended error information, use **GetLastError**.)

**Comments:** A common error is that no record exists describing the movie with the name given.

An *openFlags* value of `0` means that the movie is to be opened with sharing allowed. To open a movie with exclusive access, use `PdrOpenExclusive` as the *openFlags* value.

## PdrReadExtension

**#include "tekpdr.h"**

**BOOL PdrReadExtension ( const PdrExtensionInfo*** *pInfo*,
**void**** *dataPtr*, );

This functions returns a data buffer that corresponds to the given extension.

**Parameters:**

*pInfo:* Pointer to PdrExtensionInfo structure. All fields in this structure must be filled in prior to the call to **PdrReadExtension**. Empty strings in this structure are interpreted as “don't care.”

*dataPtr:* A pointer to the buffer that is allocated to contain the extension data.

**Return Value:** The function returns TRUE if the extension data can be successfully returned. Otherwise the function returns FALSE. To get extended error information use the **GetLastError** function.

**Comments:** If successful, a malloced buffer is returned, pointed to by **dataPtr*. This buffer must be freed by **PdrFreeExtension**.

## PdrRenameMovie

**#include "tekpdr.h"**

**BOOL PdrRenameMovie**(**ConnectHandle** *connHandle*,
**const CHAR*** *complexCurrentName*,
**const CHAR*** *complexNewName*)

This function changes the name of the existing movie to the *complexMovieName* given by the input string.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.
*complexCurrentName:* The current name of the movie to be renamed.
*complexNewName:* The new name of the movie.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The movie to be renamed must not be open unless you are using the *PdrRenameOpenMovie* command. Movies cannot be renamed if they are being used by any process or thread.

## PdrRenameOpenMovie

**#include "tekpdr.h"**

**BOOL PdrRenameOpenMovie**(**ConnectHandle** *connHandle,*
**MovieToken** *movie,*
**const CHAR*** *pcomplexNewName*)

This function changes the name of the existing open movie to the *complexMovieName* given by the input string.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.
*movie:* The token of the open movie whose name is to be modified.
*pcomplexNewName:* The new name of the open movie.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## PdrRestoreMovieDescription

**#include "tekpdr.h"**

**BOOL PdrRestoreMovieDescription**(**ConnectHandle** *connHandle,*
**const CHAR*** *complexMovieName,*
**UINT** *flags*)

This function searches for a file with the name *complexMovieName*.cmf in the media file system and uses the information in that file to restore a common movie format description for the named movie.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*complexMovieName:* The name of the movie whose description is to be restored.

*flags:* The value which controls the restoration process. Only one of the following values may be set:

- PDR_REPLACE_EXISTING
- PDR_UPDATE_EXISTING
- PDR_RESTORE_USERDATA_ONLY

In addition to one of these, PDR_NEW_MEDIA_FILES and PDR_DELETE_FILE may also be set.

**PDR_REPLACE_EXISTING**. This flag causes a new version of the movie to be placed into the inventory. If that movie already exists in the inventory, it is replaced with the new version created from the *.cmf* file.

**PDR_UPDATE_EXISTING**. This flag causes the the movie to be restored only if it does not exist in the inventory.

**PDR_RESTORE_USERDATA_ONLY**. This flag value causes the user data from the .cmf file to be stored with the movie. Existing values for the user data are replaced with the values in the *.cmf* file.

**PDR_NEW_MEDIA_FILES**. This flag creates new media filenames. It is the responsibility of the application to store the appropriate information in the media files. (The Archive Library Service uses this technique to keep from overwriting existing media files when restoring a movie.) If the media files for a movie already exist, then the names in the .cmf file are probably the correct names to use, instead of new names.

**PDR_DELETE_FILE**. This flag causes the *.cmf* file to be deleted after the restore operation.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The function of resolving movie information in the database is usually performed either automatically, or using a utility program such as regenCMF. The program shouldn't need to use this function.

## PdrSaveMovie

**#include "tekpdr.h"**

**WaitToken PdrSaveMovie(const CHAR*** *complexMovieName,*
**UINT** *numResources,*
**ResourceHandle*** *resourceList,*
**INT** *startField,*
**INT** *endField,*
**PdrCopyType** *type*)

This function creates a new movie description in the common movie format from the media files attached to the resources of the Port Timeline. The area of the timeline between *startField* and *endField* for each resource in the *resourceList* of a codec or recorder type is converted into the movie specification with the given name.

**Parameters:**

*complexMovieName:* The name for the newly created movie.

*numResources:* The number of items in the *resourceList*.

*resourceList:* An array of ResourceHandles for the Port Timeline, some of which are codec and recorders to which the media files of the movie are attached.

*startField:* The first field on the timeline which will be included in the new movie.

*endField:* The field just after the last field on the timeline which will be included in the new movie.

*type:* The manner in which the new movie should be formed: **PdrShareMedia**, **PdrRenderedMedia**, or **PdrExactMedia**.

**Return Value:** This function returns a non-zero WaitToken if the initializing operations were successful; otherwise, the function returns 0. The extended error information can be obtained from the **GetLastError** function.

**Comments:** If the PdrCopyType is PdrRenderedMedia, or PdrExactMedia, the operation requires that the embedded software make copies of media files. This operation will compete for the resources with play and record operations and will require some time to complete. For this reason, the operation is started as an asynchronous background task. This function returns immediately after initializing the background task. To determine the status of the background task or to cancel the background task, the application uses the WaitToken returned by *PdrSaveMovie*.

With **PdrRenderedMedia**, only the fields of each media file that participate in the movie are copied into a new file at the same relative position as on the timeline.

With **PdrExactMedia**, each file that participates in the movie on the timeline is copied completely.

**See Also:** *PdrGetWaitOpStatus* on page 165,
*PdrTerminateWaitOperation* on page 193.

## PdrSetClipIn

**#include "tekpdr.h"**

**BOOL PdrSetClipIn**(**MovieToken** *movie,*
**INT** *fieldNum*)

This function changes the in-point of all media segments of the specified movie. For a simple movie clip, the function will lengthen or shorten the movie by the changing the in-point. The new in-point becomes 0 on the movie's relative timeline. The Movie MarkIn value is also set to this same field number.

**Parameters:**

*movie:* The name of the movie whose in-point is modified.

*fieldNum:* The field in which the modification takes place.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** For complex movies with multiple media segments per track, the results are ill-defined. Each media segment in the movie will have the in-point set to the value `fieldNum`, which is a point relative to the first field in the media file.

## PdrSetClipOut

**#include "tekpdr.h"**

**BOOL PdrSetClipOut**(**MovieToken** *movie*,
**INT** *fieldNum*)

This function changes the out-point of all media segments of the specified movie. For a simple movie clip, the function will change the length of the movie proportional to the change in the value of the out-point field number. The Movie MarkOut value is also set to this same field number.

**Parameters:**

*movie:* The name of the movie whose out-point is changed.

*fieldNum:* The field in which the modification takes place.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** For complex movies with multiple media segments per track, the results are ill-defined. Each media segment in the movie will have the out-point set to the value `fieldNum`, which is a point relative to the first field in the media file.

## PdrSetCurrentDataset

**#include "tekpdr.h"**

**BOOL PdrSetCurrentDataset**(**ConnectHandle** *connHandle*,
**const CHAR*** *newValue*)

This function changes the value of the current dataset to the new values passed as a parameter to the call.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*newValue:* The changed value of the dataset.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The dataset part must be a valid dataset. It must be one of the file systems that exists on the Profile system.

If the function succeeds, future expansions of complex movie names will use the new dataset name.

The dataset name includes its terminating colon (:) character.

## PdrSetCurrentGroup

**#include "tekpdr.h"**

**BOOL PdrSetCurrentGroup**(**ConnectHandle** *connHandle*,
**const CHAR*** *newValue*)

This function changes the value of the current group to the new value passed as a parameter to the call.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*newValue:* The changed value of the group name.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** If the function succeeds, future expansions of complex movie names will use the new group name.

The name of the new group should ***not*** be punctuated with a leading or trailing slash (/) character.

## PdrSetMediaIn

**#include "tekpdr.h"**

**BOOL PdrSetMediaIn**(**MediaToken** *media*,
**INT** *fieldNum*)

This function changes the in-point for a media file that is part of a movie. This form of editing determines how much of the media file will participate on the port timeline.

**Parameters:**

*media:* A reference to the file whose in-point is changed.

*fieldNum:* The field number that will be the in-point of the edit.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The in-point is defined as the first field in a media file that is to be recorded or played. The fields in a media file are numbered from 0. Media files are addressable at the field level.

The in-point must be positive and it must be less than or equal to the out-point value. If the in-point is the same as the out-point value, no fields in the file participate in the record or playback operations.

## PdrSetMediaMarks

**#include "tekpdr.h"**

**BOOL PdrSetMediaMarks**(**MediaToken** *media*,
**INT** *inFieldNum*,
**INT** *outFieldNum*)

This function changes the in-point and out-point for a media file that is part of a movie. This form of editing determines how much of the media file will participate on the port timeline.

**Parameters:**

| | |
|---|---|
| *media:* | A reference to the file whose in-point and out-point are changed. |
| *inFieldNum:* | The field number that will be the in-point of the edit. |
| *outFieldNum:* | The field number that will be the out-point of the edit. |

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The in-point is defined as the first field in a media file that is to be recorded or played. The out-point is defined as the first field after the last field that is to be recorded or played in a media file. The fields in a media file are numbered from 0. Media files are addressable at the field level.

The in-point and out-point must be positive and the in-point must be less than or equal to the out-point value. If the in-point is the same as the out-point value, no fields in the file participate in the record or playback operations.

## PdrSetMediaOut

**#include "tekpdr.h"**

**BOOL PdrSetMediaOut**(**MediaToken** *media,*
**INT** *fieldNum*)

This function changes the out-point of a media file that is part of a movie. This form of editing determines how much of the media file will participate on the port timeline.

**Parameters:**

*media:* A reference to the file whose out-point is changed.

*fieldNum:* The field number that will be the out-point of the edit.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The out-point is defined as the first field after the last field that is to be recorded or played in a media file. The fields in a media file are numbered from 0. Media files are addressable at the field level.

The out-point must be positive and it must be greater than or equal to the in-point value. If the out-point is the same as the in-point value, no fields in the file participate in the record or playback operations.

## PdrSetMovieLocked

**#include "tekpdr.h"**

**BOOL PdrSetMovieLocked(MovieToken** *movie*)

This function changes the attributes of the movie that has been opened and has the given movie token.

**Parameters:**

*movie:* The token for the movie which is to be locked so that it may not be changed in any way.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The movie attributes will become PdrReadOnly, PdrControlRO, and PdrLocked. The default attributes of a new movie are ReadWrite and Unlocked.

A movie that has the PdrLocked attribute cannot be changed. It will always play exactly the same. It cannot be over-recorded, nor can its media's in-point or out-point be changed. It cannot have media added or deleted, nor can it be renamed.

To ensure that the movie cannot be over-recorded, the media files of the movie are set to ReadOnly. If these files are shared, they may cause a side effect to any movie with which they are shared. Movies that share the ReadOnly media files will have their attributes set to PdrReadOnly, but they will not have the PdrControlRO attribute.

## PdrSetMovieMarkIn

**#include "tekpdr.h"**

**INT PdrSetMovieMarkIn(MovieToken** *movie,*
**INT** *nFieldNum***)**

This function sets the movie level IN value to nFieldNum for an open movie.

**Parameters:**

| | |
|---|---|
| *movie:* | The token for the movie for which an IN mark value needs to be set. |
| *nFieldNum:* | The value to which the IN value of a movie will be set. |

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The movie marks are set during “Record” by the update thread in TekVdr. They are also set during a trim operation by VdrPanel and by the *PdrSetClipIn* function. The Movie Mark values are used as reference values, and do not directly affect how a movie is attached to the timeline.

## PdrSetMovieMarkOut

**#include "tekpdr.h"**

**INT PdrSetMovieMarkOut**(**MovieToken** *movie,*
*int nFieldNum*)

This function sets the movie level IN value to nFieldNum for an open movie.

**Parameters:**

| | |
|---|---|
| *movie:* | The token for the movie for which an OUT mark value needs to be set. |
| *nFieldNum:* | The value to which the OUT value of a movie will be set. |

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The movie marks are set during “Record” by the update thread in TekVdr. They are also set during a trim operation by VdrPanel and by the *PdrSetClipOut* function.

## PdrSetMovieMarks

**#include "tekpdr.h"**

**INT PdrSetMovieMarks**(**ConnectHandle** *hconn*,
**const CHAR*** *movieName*,
**INT** *nInFieldNum*,
**INT** *nOutFieldNum*,
**PdrMarks** *setWhichMarks*)

This function sets the movie level marks without requiring that the movie be opened.

**Parameters:**

| | |
|---|---|
| *hConn:* | The connection handle for the path from the library to the desired Profile. The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *movieName:* | The name of the movie for which the mark information is to be set. |
| *nInFieldNum:* | The field number to which the movie level mark IN value will be set (if an IN value will be set). |
| *nOutFieldNum:* | The field number to which the movie level mark OUT value will be set (if an OUT value will be set). |
| *setWhichMarks:* | This parameter specifies which marks should be set. It can specify **PdrMarkIn**, **PdrMarkOut**, or **PdrMarkBoth**. |

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** For an open movie, a single mark can be set using the *PdrSetMovieMarkIn* or *PdrSetMovieMarkOut* functions.

## PdrSetMovieReadOnly

**#include "tekpdr.h"**

**BOOL PdrSetMovieReadOnly**(**MovieToken** *movie*)

This function changes the attributes of the movie that has been opened and has the given movie token. The movie attributes will become PdrReadOnly and PdrControlRO.

**Parameters:**

*movie:* The open movie to set to ReadOnly.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The default attributes of a new movie are ReadWrite. A movie that has the PdrReadOnly attribute cannot be recorded, nor can the media files be changed, though it may still be edited. For example, it can have its media's in-point or out-point changed, it can have media added or deleted, and it can be renamed.

To ensure that the movie cannot be over-recorded, the media files of the movie are set to ReadOnly. If these files are shared, they may cause a side effect to any movie with which they are shared. Movies that share the ReadOnly media files will have their attributes set to PdrReadOnly, but they will not have the PdrControlRO attribute.

## PdrSetMovieReadWrite

**#include "tekpdr.h"**

**BOOL PdrSetMovieReadWrite**(**MovieToken** *movie*)

This function changes the attributes of a movie that has been opened and has the given movie token. The movie attributes will not contain PdrReadOnly, PdrControlRO, and PdrLocked bits when it is ReadWrite.

**Parameters:**

*movie:* The open movie to set to ReadWrite.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The default attributes of a new movie are ReadWrite and Unlocked. A movie that has the PdrLocked attribute cannot be changed. It must be unlocked before the attributes can be changed to ReadWrite. Likewise, a movie without the PdrControlRO attribute cannot be made ReadWrite; it has received the PdrReadOnly attribute passively because a media file that it shares is also used in a movie which has been explicitly set to PdrReadOnly.

When a movie is set to ReadWrite, its PdrControlRO attribute will be removed (set to 0.) It may, or may not have its PdrReadOnly bit removed. It will remain PdrReadOnly if a shared media file has been set to PdrReadOnly by another movie.

## PdrSetMovieUnlocked

**#include "tekpdr.h"**

**BOOL PdrSetMovieUnlocked(MovieToken** *movie*)

This function changes the attributes of the movie that has been opened and has the given movie token. The movie attributes will become PdrReadOnly and PdrControlRO. The PdrLocked bit will be cleared.

**Parameters:**

*movie:* The token for the movie which is to be unlocked so that it may be edited.

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** This function allows a movie to be edited but prevents a movie from being over-recorded. For example, it can have its media's in-point or out-point changed, it can have media added or deleted, and it can be renamed.

The default attributes of a new movie are ReadWrite and Unlocked. A movie that has the PdrLocked attribute cannot be changed. It must be successfully unlocked before the attributes can be changed to ReadWrite. Likewise, a movie without the PdrControlRO attribute cannot be made ReadWrite; it has received the PdrReadOnly attribute passively because a media file that it shares is also used in a movie which has been explicitly set to PdrReadOnly.

## PdrSetRegistry

**#include "tekpdr.h"**

**BOOL PdrSetRegistry**(**ConnectHandle** *connHandle*,
**PdrRegistry*** *regValues*)

This sets into the NT registry the configuration values that change the maximum size of the movie list, the referenced media list, and the default space of a movie description.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*regValues:* The address of a PdrRegistry structure that contains the new registry values. The PdrRegistry structure is defined as:

```
typedef struct
{
   UINT MaxMovies;/* Maximum number of PdrMovies */
   UINT MaxMediaRef;/* Maximum number of Media References */
   UINT MinMediaDef;/* Minimum number of Media Definitions */
}
PdrRegistry;
```

**Return Value:** TRUE if the operation is successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The registry values specify the size of the movie list (with an entry for the maximum number of movies) and the size of the media reference list (with an entry for the maximum number of media references). When the actual number of movies or media references reaches ten percent of the values cached from the registry, TekPdr will log an NT event warning. In such a case, you can use this function to delete unwanted movies and media references or increase the maximum values in the registry. The new maximums will not be effective until all applications using the library have terminated. (This can be done by shutting down and restarting the system.)

The maximum movie definition value controls the default size of new movies that are created.

**See Also:** *PdrGetRegistry* on page 157.

**Example 4. Set the current PdrRegistry settings**

```
SetPdrRegistryValues()
{
   PdrRegistry RegVal;

   /* First get the current values */
   if (PdrGetRegistry(NULL, &RegVal)){
      /* increase maximum movies by 200 */
      RegVal.MaxMovies += 200;
      /* set the new value, leaving the others unchanged */
      if (PdrSetRegistry(NULL, &RegVal))
         printf("Max Movies increased to %d\n", RegVal.MaxMovies);
      else
         printf("Error ox%x setting PdrRegistry values\n",GetLastError());
```

```
    }
    else
        printf("Error ox%x getting PdrRegistry values\n",GetLastError());
}
```

## PdrSetUserData

**#include "tekpdr.h"**

**BOOL PdrSetUserData**(**MovieToken** *movie,*
**TrackToken** *track,*
**MediaToken** *mediaSegment,*
**UINT** *idTag,*
**VOID*** *value,*
**UINT** *valueLen*)

This function copies the data identified by the pointer *value* into the movie inventory and associates the data with the specific key comprised of *movie*, *track*, *mediaSegment*, and *idTag*.

**Parameters:**

| | |
|---|---|
| *movie:* | The token for the movie with which the user data is to be associated. |
| *track:* | The token for a track of the movie with which the user data is to be associated. |
| *mediaSegment:* | A media segment with which the user data is to be associated. |
| *idTag:* | An application supplied key with which to identify this user data from other user data stored with the same movie, track and media segment key. |
| *value:* | A pointer to the bytes of data that are to be saved with the movie description. |
| *valueLen:* | The number of bytes of data to be saved with the movie description. If the key identifies data already stored with the movie, it is deleted and the new data is saved. If the number of bytes of data to store is 0, the key is deleted and no data is associated with the key. |

**Return Value:** TRUE if the user data is set; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The data is associated with a specific key, which must be used when the data is retrieved. The *idTag* is comprised of a vendor part identifying the specific software vendor and an item part which the vendor has defined for identifying this data. The key identifies specific data which might be associated with a media segment of a movie. Data can be associated with a track object of the movie by using the value of *PdrNullMediaToken* as the media segment part of the key. The data can be associated with the movie itself by using 0 as the track part of the key.

The TekPdr library does not interpret or use the data associated with an object in the movie inventory. The library will return the data when requested by calling the *PdrGetUserData* function with the same key used to store the data. The library will copy the data and move it when the movie is copied and/or moved.

Data that are associated with an object of a movie will be deleted when the object is deleted and can no longer have associated data. (Remember, tracks exist only when occupied by media segments, so the deletion of the last media segment from a track will cause an implicit deletion of the track and any data associated with it.)

**See Also:** *PdrGetUserData* on page 164.

## PdrTerminateWaitOperation

**#include "tekpdr.h"**

**BOOL** PdrTerminateWaitOperation(**WaitToken** *wait*,
**PdrWaitOpStatus*** *pStatus*)

This function closes an asynchronous operation begun with the *PdrSaveMovie* or *PdrCopyMovie* functions and gets the final status information. If the operation is not complete, it will be canceled immediately before completion which could leave files in an indeterminate state.

**Parameters:**

*wait:* A WaitToken returned by a *PdrSaveMovie* or *PdrCopyMovie* function.

*pStatus:* A pointer to a structure that will receive the final status information about the asynchronous operation.

**Return Value:** TRUE if there were no problems terminating the asynchronous operation; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The state member of the PdrWaitOpStatus structure will indicate whether the operation completed or was canceled. If the operation completed, the percent member of the PdrWaitOpStatus will be 100.

If the application knows the operation has completed because of feedback from *PdrGetWaitOpStatus*, no *PdrWaitOpStatus* structure needs to be supplied to this function. The value Null can be used.

This function must be called to retire the WaitToken so that other asynchronous operations can be initiated. The library restricts the number of asynchronous operations that are initiated but not completed. An initiation operation will fail if there are no available WaitTokens. The WaitToken will be invalid after calling this function and can no longer be used.

**See Also:** *PdrCopyMovie* on page 90,
*PdrGetWaitOpStatus* on page 165, and
*PdrSaveMovie* on page 175.

## PdrTrimClip

**#include "tekpdr.h"**

**BOOL PdrTrimClip**(**MovieToken** *movie*,
**INT** *startField*,
**INT** *endField*)

This function erases all the media files associated with a simple movie clip between the *startField* and the *endField*. The function can return disk space that was recorded beyond the fields of interest.

**Parameters:**

| | |
|---|---|
| *movie:* | A reference to the movie to be erased. |
| *startField:* | The first field of each media file to be erased. |
| *endField:* | The field beyond the last field of each media file to be erased. |

**Return Value:** This function returns an error if the movie is not a simple clip. The function will return an error if the media cannot be erased. The extended error code will be supplied by the media file system. The function will return an error if the movie is Locked or ReadOnly.

**Comments:** Media shared with other movies will not be erased.

This function does not change the media segment's in-point or out-point, or the movie length. It may change the values of the media files FirstAvailableField and/or LastAvailableField.

Playback of an area of a media file that has been erased shows as black video, silent audio, and timecode 00:00:00:00.

The *startField* and *endField* are relative to the file, and not relative to the timeline. If the clip has its media in-points set at 100, a PdrTrimClip operation with *startField* = 0 and *endField* = 100 will have no effect on the clip as played. The erasure is of unused material prior to the in-point. The in-point is the first field of the file on the timeline.

# The Pls library

The Profile Library System (TekPls) works with files as a basic unit of information. A file can be a simple stream of bytes or a multiplexed stream of video, audio, and timecodes. The library system copies files from a Profile to a tape cartridge and back. The library server does not delete files from a Profile. Management of the material on a Profile is the application's responsibility.

## PlsAddTransport

**#include "tekpls.h"**

**BOOL PlsAddTransport(PlsLibraryHandle** *hLibrary*,
**SHORT** *transportNumber*,
**SHORT** *robotAddress*,
**PlsHostNameStr** *hostName*,
**PlsDeviceAddressStr** *deviceAddress*)

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *transportNumber:* | Transport number. |
| *robotAddress:* | Address of tape cartridge robot. |
| *hostName*: | Host name. |
| *deviceAddress*: | Device address string. |

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** In earlier versions, it was necessary for the last parameter to be supplied by building the SCSI address. Now the library will build the SCSI address at the next available slot, if you pass NULL for deviceAddress.

## PlsAllocateCartridge

**#include "tekpls.h"**

**PlsCartridgeHandle PlsAllocateCartridge(PlsLibraryHandle** *hLibrary*,
**PlsCartridgeLabelStr** *cartLabel*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *cartLabel*: | Cartridge barcode label string. |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** This function returns the cartridge handle if there are no errors. Otherwise, it returns a NULL value.

## PlsAllocateTransport

**#include "tekpls.h"**

**PlsTransportHandle PlsAllocateTransport(PlsLibraryHandle** *hLibrary*,
**SHORT** *transportNumber*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

*hLibrary*: Library handle.
*transportNumber*: Transport number.
*phTransaction*: Pointer to transaction handle.

**Return Value:** This function returns the transport handle if there are no errors. Otherwise, it returns a NULL value.

## PlsArchiveClip

**#include "tekpls.h"**

**BOOL PlsArchiveClip(PlsFileHandle** *hFile*,
**INT** *archiveIn*,
**INT** *archiveOut*,
**PlsTransportHandle** *hTransport*,
**PlsFileStored** **pFileStored*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hFile*: | File handle. |
| *archiveIn*: | In-point for archiving. |
| *archiveOut*: | Out-point for archiving. |
| *hTransport*: | Transport handle. |
| *pFileStored*: | Pointer to a structure which will have information on the partition number and the cartridge label where the file was stored. |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsArchiveDataFile

**#include "tekpls.h"**

**BOOL PlsArchiveDataFile(PlsFileHandle** *hFile*,
**PlsTransportHandle** *hTransport*,
**PlsFileNameStr** *srcNTFile*,
**PlsFileStored** **pFileStored*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hFile*: | File handle. |
| *hTransport*: | Transport handle. |
| *srcNTFile*: | NT filename string. |
| *pFileStored*: | Pointer to a structure which will have information on the partition number and the cartridge label where the file was stored. |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsBackupCatalog

**#include "tekpls.h"**

**BOOL PlsBackupCatalog(PlsLibraryHandle** *hLibrary*)

**Parameters:**

*hLibrary*: Library handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsCancelCommand

**#include "tekpls.h"**

**BOOL PlsCancelCommand(PlsLibraryHandle** *hLibrary*,
**PlsTransactionHandle** *hTransaction*)

Cancels an asynchronous, concurrent operation.

**Parameters:**

*hLibrary*: Library handle.
*hTransaction:* Transaction handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** Not all the asynchronous commands issued can be canceled. Only the following commands can be canceled:

- Archive
- Restore
- CopyCartridge

## PlsCleanTransport

**#include "tekpls.h"**

**BOOL PlsCleanTransport(PlsTransportHandle** *hTransport*,
**PlsCartridgeHandle** *hCartridge*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hTransport*: | Transport handle. |
| *hCartridge*: | Cartridge handle. |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsCloseBinInfo

**#include "tekpls.h"**

**BOOL PlsCloseBinInfo(PlsBinInfoLoopHandle** *hLoop*)

**Parameters:**

*hLoop*: Bin information loop handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsCloseCartridge

**#include "tekpls.h"**

**BOOL PlsCloseCartridge**(**PlsCartridgeHandle** *hCartridge*)

**Parameters:**

*hCartridge*: Cartridge handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsCloseCartridgeInfo

**#include "tekpls.h"**

**BOOL PlsCloseCartridgeInfo**(**PlsCartInfoLoopHandle** *hLoop*)

**Parameters:**

*hLoop*: Bin information loop handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsCloseFile

**#include "tekpls.h"**

**BOOL PlsCloseFile**(**PlsFileHandle** *hFile*)

**Parameters:**

*hFile*: Filename (handle).

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsCloseFileInfo

**#include "tekpls.h"**

**BOOL PlsCloseFileInfo(PlsFileInfoLoopHandle** *hLoop*)

**Parameters:**

*hLoop*: Bin information loop handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsCloseFindHandle

**#include "tekpls.h"**

**BOOL PlsCloseFindHandle**(**PlsHandleLoopHandle** *hLoop*)

**Parameters:**

*hLoop*: Bin information loop handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsCloseLibrary

**#include "tekpls.h"**

**BOOL PlsCloseLibrary**(**PlsLibraryHandle** *hLibrary*)

This closes the specified library.

**Parameters:**

*hLibrary*: Library handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsCloseTransport

**#include "tekpls.h"**

**BOOL PlsCloseTransport(PlsTransportHandle** *hTransport*)

**Parameters:**

*hTransport*: Transport handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsCloseTransportConfig

**#include "tekpls.h"**

**BOOL PlsCloseTransportConfig(PlsTransConfigLoopHandle** *hLoop*)

This closes the tranport configuration connection.

**Parameters:**

*hLoop*: Transport configuration loop handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsConnectCartridge

**#include "tekpls.h"**

**PlsCartridgeHandle PlsConnectCartridge**(**PlsLibraryHandle** *hLibrary*,
**PlsCartridgeLabelStr** *cartLabel*)

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *cartLabel*: | Cartridge barcode label string. |

**Return Value:** This function returns the cartridge handle if there are no errors. Otherwise, it returns a NULL value.

## PlsConnectFile

**#include "tekpls.h"**

**PlsFileHandle PlsConnectFile**(**PlsLibraryHandle** *hLibrary*,
**PlsCartridgeHandle** *hCartridge*,
**SHORT** *partitionNumber*,
**PlsFileNameStr** *fileName*)

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *hCartridge*: | Cartridge handle. |
| *partitionNumber*: | The partition number. |
| *fileName*: | Filename string. |

**Return Value:** This function returns the file handle if there are no errors. Otherwise, it returns a NULL value.

## PlsConnectTransport

**#include "tekpls.h"**

**PlsTransportHandle PlsConnectTransport**(**PlsLibraryHandle** *hLibrary*,
**SHORT** *transportNumber*)

**Parameters:**

*hLibrary*: Library handle.
*transportNumber*: Transport number.

**Return Value:** This function returns the transport handle if there are no errors. Otherwise, it returns a NULL value.

## PlsCopyCartridge

**#include "tekpls.h"**

**BOOL PlsCopyCartridge(PlsCopyCartAction** *copyCartActionCode*,
**PlsTransportHandle** *src_hTransport*,
**PlsCartridgeHandle** *src_hCartridge*,
**PlsTransportHandle** *dst_hTransport*,
**PlsCartridgeHandle** *dst_hCartridge*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

*copyCartActionCode*:Copy cartridge action code. One of the following:

- *PlsCopyCartNormal*: Harden deletes, all other file copied to same partition.
- *PlsCopyCartExact*: A one-to-one copy.
- *PlsCopyCartCompact*: Harden deletes and move files to lower partitions.

*src_hTransport*: Transport handle to source transport?
*src_hCartridge*: Cartridge handle to source cartridge?
*dst_hTransport*: Transport handle to destination transport.
*dst_hCartridge*: Cartridge handle to destination cartridge.
*phTransaction*: Pointer to transaction handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsDeleteFile

**#include "tekpls.h"**

**BOOL PlsDeleteFile(PlsFileHandle** *hFile*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hFile*: | Filename (handle). |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsExportCartridge

**#include "tekpls.h"**

**BOOL PlsExportCartridge(PlsCartridgeHandle** *hCartridge*,
**PlsExportAction** *exportActionCode*,
**PlsLocationInfoStr** *locationInfo*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

*hCartridge*: Cartridge handle.

*exportActionCode*:Export action code. One of the following:

- *PlsExportActionNormal*: Export a cartridge, but don't forget about it (keep catalog entries).
- *PlsExportActionRetire*: Update cartridge directory, export it and remove entries from the catalog.
- *PlsExportActionForget.* Export cartridge, don't update cartridge directories and delete catalog entries.

*locationInfo*: Location information string. This indicates where the cartridge is stored when it is out of library.

*phTransaction*: Pointer to transaction handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsFindFirstBinInfo

**#include "tekpls.h"**

**PlsBinInfoLoopHandle PlsFindFirstBinInfo(PlsLibraryHandle** *hLibrary*,
**PlsBinInfoAction** *binInfoAction*,
**INT** *startingBin*,
**PlsBinInfo** **pBinInfo*)

**Parameters:**

*hLibrary*: Library handle.

*binInfoAction*: Bin information action code. One of the following:

- *PlsBinInfoFindAll*: Get information on all bins.
- *PlsBinInfoFindData*: Only search archive data cartridges.
- *PlsBinInfoFindCleaning*: Restrict search to cleaning cartridges.
- *PlsBinInfoFindBackup*: Get information on all bins with archive server backup cartridges.
- *PlsBinInfoFindUnformatted*: Restrict search to unformatted cartridges.
- *PlsBinInfoFindUnknown*: Restrict search to unknown cartridges.

*startingBin*: Starting bin number (location).

*pBinInfo*: Pointer to bin information structure. This structure contains following information:

A status word with the following flags:

Bin is available (magazine installed.) This does not mean that the bin is empty.

Bin can hold a cartridge and is empty.

This bin can store a library server data cartridge.

This bin can store a transport cleaning cartridge.

This bin can store an unformatted cartridge.

This bin can store a cartridge with unknown contents.

This bin contains a library server data cartridge.

This bin contains a transport cleaning cartridge.

This bin contains an unformatted cartridge.

This bin contains a cartridge with unknown contents.

Bin number (location).

Cartridge label if a tape is present.

Cartridge class this bin stores.

**Return Value:** This function returns the loop handle if there are no errors. Otherwise, it returns a NULL value.

## PlsFindFirstCartridgeInfo

**#include "tekpls.h"**

**PlsCartInfoLoopHandle PlsFindFirstCartridgeInfo(PlsLibraryHandle** *hLibrary*,
**PlsCartInfoAction** *cartInfoAction*,
**PlsCartridgeLabelStr** *startingCart*,
**PlsCartridgeStatus** **pCartStatus*)

**Parameters:**

*hLibrary*: Library handle.

*cartInfoAction*: Cartridge information action code. One of the following:

- *PlsCartInfoFindAll*: Get information on all cartridges.
- *PlsCartInfoFindLib*: Get information on cartridges in a library(not exported).
- *PlsCartInfoStorage*: Information on cartridges out of library, but in local catalog.

*startingCart*: A barcode label of the cartridge where the search should start from.

*pCartStatus*: Pointer to the cartridge status structure. The returned information includes:

A status word with the following flags:

- Cartridge is reserved
- Cartridge in use
- A library server data cartridge
- A cleaning cartridge
- A library server database backup cartridge
- Cartridge contents are unknown
- Cartridge has been formatted
- Cartridge is write-protected
- Cartridge is in library
- Cartridge is formatted for one file per partition
- Cartridge needs updating
- Cartridge is stuck in the transport

Total free space on cartridge (in megabytes)

Largest free space on this tape (in megabytes)

Partition size (in megabytes) if partitioned (or 0 if not partitioned)

Number of partitions (or 0 if not partitioned)

Cartridge label for this cartridge

A time/date stamp with the most recent of the following:

- When the cartridge was formatted
- When the cartridge was imported
- When the cartridge was exported

If reserved, owner name for this library connection

Space allocation grouping control (categories)

If out of the library, indication of physical location of the cartridge.

**Return Value:** This function returns the loop handle if there are no errors. Otherwise, it returns a NULL value.

## PlsFindFirstFileInfo

**#include "tekpls.h"**

**PlsFileInfoLoopHandle PlsFindFirstFileInfo(PlsLibraryHandle** *hLibrary*,
**PlsFileInfoAction** *fileInfoAction*,
**PlsCartridgeLabelStr** *cartLabel*,
**SHORT** *partitionNumber*,
**PlsFileNameStr** *fileName*,
**PlsFileInfo** **pFileInfo*)

**Parameters:**

*hLibrary*: Library handle.

*fileInfoAction*: File information action code. One of the following:

- *PlsFileInfoFindAll*: Get information on all files.
- *PlsFileInfoFindLib*: Get information on all files in a library.
- *PlsFileInfoFindStore*: Get information on all files in storage(not in library).
- *PlsFileInfoFindAllDeleted*: Get Information on all files even deleted ones.

*cartLabel*: Cartridge barcode label string.

*partitionNumber*: The partition number.

*fileName*: Filename string.

*pFileInfo*: Pointer to file info structure. This structure contains the following information:

A status word with the following flags:

File is read only

This file has been deleted

This file has been renamed

This is a clip file

This is a data file

Space required for the file (in megabytes)

Filename (current name space)

File type (file contents) identification

Stored clip actual in-point

Stored clip actual out-point

Number of video tracks in this file

Number of audio tracks in this file

Number of timecode tracks in this file

When was archive file (not the source file) created

When was the archived file last referenced (only available for library-based systems)

Cartridge label for tape with this file

Partition number file is in

A time/date stamp for a library-server-assisted file purge

**Return Value:** This function returns the loop handle if there are no errors. Otherwise, it returns a NULL value.

## PlsFindFirstHandle

**#include "tekpls.h"**

**PlsHandleLoopHandle PlsFindFirstHandle(PlsLibraryHandle** *hLibrary*,
**PlsHandleInfoAction** *handleInfoAction*,
**PlsHandleInfo** **handleInfo*)

**Parameters:**

*hLibrary*: Library handle.

*handleInfoAction*: The action code, which can be used to request the following:

- *PlsFindLibraryHandles* for library handles;
- *PlsFindFileHandles* for file handles;
- *PlsFindCartridgeHandles* for cartridge handles;
- *PlsFindTransportHandles* for transport handles; or
- *PlsFindTransactionHandles* for transaction handles.

*handleInfo*: Pointer to handle information structure. The handle info contains returned information for each type of handle.

For library handles:

Session name

For file handles:

Filename string

The cartridge barcode label string

The partition number

For cartridge handles:

The barcode label string

For transport handles:

The transport number

For transaction handles:

The active library handle

The event code for this transaction type

In addition, the following information is returned for each handle:

A handle type identifier

The actual handle value

**Return Value:** This function returns the loop handle if there are no errors. Otherwise, it returns a NULL value.

## PlsFindFirstTransportConfig

**#include "tekpls.h"**

**PlsTransConfigLoopHandle PlsFindFirstTransportConfig(PlsLibraryHandle** *hLibrary*,
**SHORT** *transportNumber*,
**PlsTransportConfig** **pTransportConfig)*

Returns a library's transport configuration.

**Parameters:**

*hLibrary*: Library handle.

*transportNumber*: Transport number.

*pTransportConfig:* Pointer to library transport configuration. This function returns the following for each transport:

- Numeric address for the transport in the range 0–255.
- Location of the transport in the library.
- Name of the Profile host machine.
- Transport manufacturer identification string.
- Transport model identification string.
- Transport version string.

**Return Value:** Transport status information.

## PlsFindNextBinInfo

**#include "tekpls.h"**

**BOOL PlsFindNextBinInfo**(**PlsBinInfoLoopHandle** *hLoop*,
**PlsBinInfo** **pBinInfo*)

**Parameters:**

*hLoop*: Bin information loop handle.

*pBinInfo*: Pointer to bin information structure. This structure contains following information:

A status word with the following flags:

Bin is available (magazine installed.) This does not mean that the bin is empty.

Bin can hold a cartridge and is empty.

This bin can store a library server data cartridge.

This bin can store a transport cleaning cartridge.

This bin can store an unformatted cartridge.

This bin can store a cartridge with unknown contents.

This bin contains a library server data cartridge.

This bin contains a transport cleaning cartridge.

This bin contains an unformatted cartridge.

This bin contains a cartridge with unknown contents.

Bin number (location).

Cartridge label if a tape is present.

Cartridge class this bin stores.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsFindNextCartridgeInfo

**#include "tekpls.h"**

**BOOL PlsFindNextCartridgeInfo**(**PlsCartInfoLoopHandle** *hLoop*,
**PlsCartridgeStatus** **pCartStatus*)

**Parameters:**

*hLoop*: Bin information loop handle.

*pCartStatus*: Pointer to the cartridge status structure. The returned information includes:

A status word with the following flags:
- Cartridge is reserved
- Cartridge in use
- A library server data cartridge
- A cleaning cartridge
- A library server database backup cartridge
- Cartridge contents are unknown
- Cartridge has been formatted
- Cartridge is write-protected
- Cartridge is in library
- Cartridge is formatted for one file per partition
- Cartridge needs updating
- Cartridge is stuck in the transport

Total free space on cartridge (in megabytes)

Largest free space on this tape (in megabytes)

Partition size (in megabytes) if partitioned (or 0 if not partitioned)

Number of partitions (or 0 if not partitioned)

Cartridge label for this cartridge

A time/date stamp with the most recent of the following:
- When the cartridge was formatted
- When the cartridge was imported
- When the cartridge was exported

If reserved, owner name for this library connection

Space allocation grouping control (categories)

If out of the library, indication of physical location of the cartridge.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsFindNextFileInfo

**#include "tekpls.h"**

**BOOL PlsFindNextFileInfo(PlsFileInfoLoopHandle** *hLoop*,
**PlsFileInfo** **pFileInfo*)

**Parameters:**

*hLoop*: Bin information loop handle.

*pFileInfo*: Pointer to file info structure. This structure contains the following information:

A status word with the following flags:

- File is read only
- This file has been deleted
- This file has been renamed
- This is a clip file
- This is a data file

Space required for the file (in megabytes)

Filename (current name space)

File type (file contents) identification

Stored clip actual in-point

Stored clip actual out-point

Number of video tracks in this file

Number of audio tracks in this file

Number of timecode tracks in this file

When was archive file (not the source file) created

When was the archived file last referenced (only available for library-based systems)

Cartridge label for tape with this file

Partition number file is in

A time/date stamp for a library-server-assisted file purge

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsFindNextHandle

**#include "tekpls.h"**

**BOOL PlsFindNextHandle(PlsHandleLoopHandle** *hLoop*,
**PlsHandleInfo** **handleInfo*)

**Parameters:**

*hLoop*: Bin information loop handle.

*handleInfo*: Pointer to handle information structure. The handle info contains returned information for each type of handle.

For library handles:

- Session name

For file handles:

- Filename string
- The cartridge barcode label string
- The partition number

For cartridge handles:

- The barcode label string

For transport handles:

- The transport number

For transaction handles:

- The active library handle
- The event code for this transaction type

In addition, the following information is returned for each handle:

- A handle type identifier
- The actual handle value

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsFindNextTransportConfig

**#include "tekpls.h"**

**BOOL PlsFindNextTransportConfig(PlsTransConfigLoopHandle** *hLoop*,
**PlsTransportConfig** **pTransportConfig*)

**Parameters:**

*hLoop*: Transport configuration loop handle.
*pTransportConfig*:Pointer to transport configuration data structure.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsFormatCartridge

**#include "tekpls.h"**

**BOOL PlsFormatCartridge**(**PlsLibraryHandle** *hLibrary*,
**PlsTransportHandle** *hTransport*,
**PlsFormatCartAction** *formatCartAction*,
**PlsCartridgeLabelStr** *cartLabel*,
**PlsCategoryStr** *category*,
**INT** *masterDirSize_Mbytes*,
**INT** *partitionSize_Mbytes*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

*hLibrary*: Library handle.
*hTransport*: Transport handle.
*formatCartAction*:Format action code, which can be one of these:

- PlsFormatCartNormal: Normal format.
- PlsFormatCartSingle: Force only one file per partition mode.

*cartLabel*: Cartridge barcode label string.
*category*: category string. Can be passed NULL.
*masterDirSize_Mbytes*:The master directory size in megabytes.
*partitionSize_Mbytes*:The data partition size in megabytes.
*phTransaction*: Pointer to transaction handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetAnyEvent

**#include "tekpls.h"**

**BOOL PlsGetAnyEvent**(**PlsLibraryHandle** *hLibrary*,
**PlsEvent** **pEvent*)

Returns information on one asynchronous event or concurrent command completion.

**Parameters:**

*hLibrary*: Library handle.

*pEvent*: Pointer to event's data structure. The event types include:

- a no-event-queued indicator.
- cartridge import (implicit or explicit), export, delete all, format events.
- archive file creation, deletion, and rename notifications.
- transport on-line/off-line (add/remove) events.
- concurrent command completion events.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetAsynchEvent

**#include "tekpls.h"**

**BOOL PlsGetAsynchEvent(PlsLibraryHandle** *hLibrary*,
**PlsEvent** **pEvent*)

Returns events other than command completion events.

**Parameters:**

*hLibrary*: Library handle.

*pEvent*: Pointer to event's data structure. The event types include:

- a no-event-queued indicator.
- cartridge import (implicit or explicit), export, delete all, format events.
- archive file creation, deletion, and rename notifications.
- transport on-line/off-line (add/remove) events.
- concurrent command completion events.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetBackupDir

**#include "tekpls.h"**

**BOOL PlsGetBackupDir**(**PlsLibraryHandle** *hLibrary*,
**PlsBackupPathStr** *path*,
**PlsBackupDriveStr** *drive*)

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *path:* | Returned path of backup directory. |
| *drive:* | Returned drive of backup directory. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetCartDescription

**#include "tekpls.h"**

**BOOL PlsGetCartDescription**(**PlsCartridgeHandle** *hCartridge*,
**PlsCartDescriptionStr** *cartDescription*)

**Parameters:**

| | |
|---|---|
| *hCartridge*: | Cartridge handle. |
| *cartDescription*: | Cartridge description string. The value returned is either user-supplied or application-supplied data. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetCartridgeConfig

**#include "tekpls.h"**

**BOOL PlsGetCartridgeConfig(PlsCartridgeHandle** *hCartridge*,
**PlsCartridgeConfig** **pCartConfig*)

**Parameters:**

*hCartridge*: Cartridge handle.

*pCartConfig*: Pointer to cartridge configuration structure. The returned configuration data includes the following information:

- Maximum possible space (in megabytes)
- Maximum possible partitions
- Cartridge class identification string
- Cartridge manufacturer, if known
- The cartridge model

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetCartridgeStatus

**#include "tekpls.h"**

**BOOL PlsGetCartridgeStatus(PlsCartridgeHandle** *hCartridge*,
**PlsCartridgeStatus** **pCartStatus*)

**Parameters:**

*hCartridge*: Cartridge handle.

*pCartStatus*: Pointer to the cartridge status structure. The returned information includes:

A status word with the following flags:
- Cartridge is reserved
- Cartridge in use
- A library server data cartridge
- A cleaning cartridge
- A library server database backup cartridge
- Cartridge contents are unknown
- Cartridge has been formatted
- Cartridge is write-protected
- Cartridge is in library
- Cartridge is formatted for one file per partition
- Cartridge needs updating
- Cartridge is stuck in the transport

Total free space on cartridge (in megabytes)

Largest free space on this tape (in megabytes)

Partition size (in megabytes) if partitioned (or 0 if not partitioned)

Number of partitions (or 0 if not partitioned)

Cartridge label for this cartridge

A time/date stamp with the most recent of the following:
- When the cartridge was formatted
- When the cartridge was imported
- When the cartridge was exported

If reserved, owner name for this library connection

Space allocation grouping control (categories)

If out of the library, indication of physical location of the cartridge.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetClipSize

**#include "tekpls.h"**

**BOOL PlsGetClipSize**(**PlsFileHandle** *hFile*,
**INT** *inPoint*,
**INT** *outPoint*,
**PlsTransportHandle** *hTransport*,
**INT** **pFileSize*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hFile*: | Filename (handle). |
| *inPoint*: | In-point for clip size calculation. |
| *outPoint*: | Out-point for clip size calculation. |
| *hTransport*: | Transport handle. |
| *pFileSize*: | Pointer to returned file size (n megabytes). |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetCommandEvent

**#include "tekpls.h"**

**BOOL PlsGetCommandEvent**(**PlsLibraryHandle** *hLibrary*,
**PlsTransactionHandle** *hTransaction*,
**PlsEvent** **pEvent*)

Allows an application to retrieve command completion events.

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *hTransaction:* | Transaction handle. |
| *pEvent*: | Pointer to event's data structure. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetEventMask

**#include "tekpls.h"**

**BOOL PlsGetEventMask(PlsLibraryHandle** *hLibrary*,
**UINT** **pEventMask*)

**Parameters:**

*hLibrary*: Library handle.

*pEventMask*: Pointer to library server event mask. The mask notifies which classes of events are enabled/disabled. The following are classes of events:

- Cartridge import, export, format, and delete all file events
- File add, delete, and rename events
- Transport on-line/off-line events
- Extended data for import, export, and format on command completion

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetFileDescription

**#include "tekpls.h"**

**BOOL PlsGetFileDescription(PlsFileHandle** *hFile*,
**PlsFileDescriptionStr** *fileDescription*)

**Parameters:**

| | |
|---|---|
| *hFile*: | Filename (handle). |
| *fileDescription*: | A string associated with each file. This gives descriptive or control data associated with file. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetLibraryConfig

**#include "tekpls.h"**

**BOOL PlsGetLibraryConfig(PlsLibraryHandle** *hLibrary*,
**PlsLibraryConfig** **pLibraryConfig*)

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *pLibraryConfig*: | Pointer to library configuration structure. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetLibraryStatus

**#include "tekpls.h"**

**BOOL PlsGetLibraryStatus(PlsLibraryHandle** *hLibrary*,
**PlsLibraryStatus** **pLibraryStatus*)

Returns library status.

**Parameters:**

*hLibrary*: Library handle.

*pLibraryStatus:* Pointer to library status data structure. Returns the following:

- status word with flags for: (1) library is operational; (2) library robot is active.
- number of empty bins.
- number of bins with archive tapes.
- number of bins with other tapes.
- number of available robots (not in use)
- number of empty entry ports
- number of empty exit ports
- number of available transports
- device-specific information (32 bytes)

Information for the Exabyte 480 library only:

- cartridge handler moves
- cartridge handler retries
- total scans
- scan retires
- scan failures
- entry/exit port cycles

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetLocationString

**#include "tekpls.h"**

**BOOL PlsGetLocationString(PlsCartridgeHandle** *hCartridge*,
**PlsLocationInfoStr** *locationInfo*)

**Parameters:**

| | |
|---|---|
| *hCartridge*: | Cartridge handle. |
| *locationInfo*: | Location information string. This indicates where the cartridge is stored when it is out of library. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetMajorVersion

**#include "tekpls.h"**

**INT PlsGetMajorVersion**(**ConnectHandle** *hConnection*)

Returns the major PLS version number. For example, the major number in version 1.23 is **1**.

**Parameters:**

*hConnection:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the major PLS version number.

## PlsGetMinorVersion

**#include "tekpls.h"**

**INT PlsGetMinorVersion**(**ConnectHandle** *hConnection*)

Returns the minor PLS version number. For example, the minor number in version 1.23 is **23**.

**Parameters:**

*hConnection:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the minor PLS version number.

## PlsGetModes

**#include "tekpls.h"**

**BOOL PlsGetModes**(**PlsLibraryHandle** *hLibrary*,
**PlsModes** **pModes*)

Returns the current current library operating modes.

**Parameters:**

*hLibrary*: Library handle.

*pModes*: Returned operating modes. The following mode variables are supported, a word with single-bit modes:

- a bit that enables transport queuing.
- a bit that enables cartridge queuing.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetPartitionMap

**#include "tekpls.h"**

**BOOL PlsGetPartitionMap**(**PlsCartridgeHandle** *hCartridge*,
**SHORT** *startingParition*,
**SHORT** *numberOfPartitions*,
**PlsPartMap** **partMap*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

*hCartridge*: Cartridge handle.
*startingParition*: The starting partition number.
*numberOfPartitions*:The number of values to be returned.
*partMap:* Where a cartridge's free space resides.
*phTransaction*: Pointer to transaction handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetPath

**#include "tekpls.h"**

**BOOL PlsGetPath**(**PlsLibraryHandle** *hLibrary*,
**PlsPathStr** *path*)

Returns current directory path.

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *path:* | Returned current directory path. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetStatusCommand

**#include "tekpls.h"**

**BOOL PlsGetStatusCommand**(**PlsLibraryHandle** *hLibrary*,
**PlsCmdStatus** **pCommandStatus*,
**PlsTransactionHandle** *hTransaction*)

**Parameters:**

*hLibrary*: Library handle.
*pCommandStatus*: Pointer to command status data structure.
*hTransaction*: Transaction handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetTimeDate

**#include "tekpls.h"**

**BOOL PlsGetTimeDate(PlsLibraryHandle** *hLibrary*,
**FILETIME** **pTimeDate*)

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *pTimeDate*: | File timestamp. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsGetTransportStatus

**#include "tekpls.h"**

**BOOL PlsGetTransportStatus(PlsTransportHandle** *hTransport*,
**PlsTransportStatus** **pTransportStatus*)

**Parameters:**

*hTransport*: Transport handle.

*pTransportStatus*: Pointer to the transport status structure. The returned information includes:

A status word with the following flags:

- Transport is reserved
- Transport in operational
- Transport in in use
- Transport needs cleaning
- Transport contains a tape

The transport logical address

Who has reserved the transport. (This is the session name string from a PlsOpenLibrary.)

Cartridge label if a cartridge is loaded

Device-specific data (32 bytes)

Device-specific information for the Exabyte Mammoth tape transport

Soft (recovered) write errors

Soft read errors

Megabytes written to the tape

Megabytes read from the tape

Hard (unrecovered) write errors

Hard read errors

Device-specific information for the Ampex DST 312 tape transport including:

- Soft (recovered) write errors
- Soft read errors
- Hard (unrecovered) write errors
- Hard read errors

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsHouseKeeping

**#include "tekpls.h"**

**BOOL PlsHouseKeeping(PlsLibraryHandle** *hLibrary*,
**INT** *reserved1*,
**PlsHouseKeepingInfoStr** *reserved2*,
**PlsTransactionHandle** **phTransaction*)

Performs system management functions, such as removing entries from the cache database that are no longer needed.

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *reserved1:* | Reserved for future use (should be coded 0 [zero]). |
| *reserved2:* | Reserved for future use. |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsImportCartridge

**#include "tekpls.h"**

**BOOL PlsImportCartridge**(**PlsLibraryHandle** *hLibrary*,
**PlsImportAction** *importActionCode*,
**PlsCartridgeLabelStr** *cartLabel*,
**PlsLocationInfoStr** *locationInfo*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

*hLibrary*: Library handle.

*importActionCode*:Import action code. One of the following:

- *PlsImportActionAccept*: Import cartridge and assume bar code label correctly identifies tape.
- *PlsImportActionVerify*: Import cartridge and make simple checks to verify contents of tape.
- *PlsImportActionInventory*: Import cartridge and read cartridge directories.
- *PlsImportActionUnformatted*: Import an unformatted tape.
- *PlsImportActionCleaning*: Import a cleaning cartridge.
- *PlsImportActionBackup*: Import an archive server database backup cartridge.

*cartLabel*: Cartridge barcode label string.

*locationInfo*: Location information string. This indicates where the cartridge is stored when it is out of library.

*phTransaction*: Pointer to transaction handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsImportLoadCartridge

**#include "tekpls.h"**

**BOOL PlsImportLoadCartridge**(**PlsLibraryHandle** *hLibrary*,
**PlsTransportHandle** *hTransport*,
**PlsImportAction** *importActionCode*,
**PlsCartridgeLabelStr** *cartLabel*,
**PlsLocationInfoStr** *locationInfo*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

*hLibrary*: Library handle.

*hTransport*: Transport handle.

*importActionCode*:Import action code. One of the following:

- *PlsImportActionAccept*: Import cartridge and assume bar code label correctly identifies tape.
- *PlsImportActionVerify*: Import cartridge and make simple checks to verify contents of tape.
- *PlsImportActionInventory*: Import cartridge and read cartridge directories.
- *PlsImportActionUnformatted*: Import an unformatted tape.
- *PlsImportActionCleaning*: Import a cleaning cartridge.
- *PlsImportActionBackup*: Import an archive server database backup cartridge.

*cartLabel*: Cartridge barcode label string.

*locationInfo*: Location information string. This indicates where the cartridge is stored when it is out of library.

*phTransaction*: Pointer to transaction handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsInventoryCartridge

**#include "tekpls.h"**

**BOOL PlsInventoryCartridge(PlsTransportHandle** *hTransport*,
**PlsCartridgeHandle** *hCartridge*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hTransport*: | Transport handle. |
| *hCartridge*: | Cartridge handle. |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsInventoryLibrary

**#include "tekpls.h"**

**BOOL PlsInventoryLibrary**(**PlsLibraryHandle** *hLibrary*,
**PlsTransactionHandle** **phTransaction*)

Verify and correct the cartridge inventory of a specific library.

**Parameters:**

*hLibrary*: Library handle.
*phTransaction:* Pointer to transaction handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsLoadTransport

**#include "tekpls.h"**

**BOOL PlsLoadTransport(PlsTransportHandle** *hTransport*,
**PlsCartridgeHandle** *hCartridge*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hTransport*: | Transport handle. |
| *hCartridge*: | Cartridge handle. |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsOpenLibrary

**#include "tekpls.h"**

**PlsLibraryHandle PlsOpenLibrary(ConnectHandle** *hConnection*,
**PlsLibraryNameStr** *libraryName*,
**PlsSessionNameStr** *sessionName*,
**SHORT** *majorVersion*,
**SHORT** *minorVersion*)

**Parameters:**

| | |
|---|---|
| *hConnection:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *libraryName*: | For future use. |
| *sessionName*: | Session name identifies who owns a resource that returns status commands. |
| *majorVersion*: | Major version number. |
| *minorVersion*: | Minor version number. |

**Return Value:** Returns a connection handle for the library.

## PlsRemoveTransport

**#include "tekpls.h"**

**BOOL PlsRemoveTransport**(**PlsLibraryHandle** *hLibrary*,
**SHORT** *transportNumber*,
**PlsTransactionHandle** **phTransaction*)

Removes the specified transport from the library server control files.

**Parameters:**

*hLibrary*: Library handle.
*transportNumber:* Transport number.
*phTransaction:* Pointer to transaction handle.

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsRenameFile

**#include "tekpls.h"**

**BOOL PlsRenameFile(PlsFileHandle** *hFile*,
**PlsFileNameStr** *newFileName*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hFile*: | Filename (handle). |
| *newFileName*: | New filename string. |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsRestore

**#include "tekpls.h"**

**BOOL PlsRestore**(**PlsFileHandle** *hFile*,
**INT** *restoreIn*,
**INT** *restoreOut*,
**PlsTransportHandle** *hTransport*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hFile*: | Filename (handle). |
| *restoreIn*: | In-point for restoring. |
| *restoreOut*: | Out-point for restoring. |
| *hTransport*: | Transport handle. |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsRestoreDataFile

**#include "tekpls.h"**

**BOOL PlsRestoreDataFile**(**PlsFileHandle** *hFile*,
**PlsTransportHandle** *hTransport*,
**PlsFileNameStr** *destNTFile*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hFile*: | Filename (handle). |
| *hTransport*: | Transport handle. |
| *destNTFile*: | NT filename string. |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsSetBackupDir

**#include "tekpls.h"**

**BOOL PlsSetBackupDir(PlsLibraryHandle** *hLibrary*,
**PlsBackupPathStr** *path*,
**PlsBackupDriveStr** *drive*)

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *path:* | Path for backup directory. |
| *drive:* | Drive for backup directory. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsSetCartDescription

**#include "tekpls.h"**

**BOOL PlsSetCartDescription**(**PlsCartridgeHandle** *hCartridge*,
**PlsCartDescriptionStr** *cartDescription*)

**Parameters:**

| | |
|---|---|
| *hCartridge*: | Cartridge handle. |
| *cartDescription*: | Cartridge description string. The value returned is either user-supplied or application-supplied data. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsSetEventMask

**#include "tekpls.h"**

**BOOL PlsSetEventMask**(**PlsLibraryHandle** *hLibrary*,
**UINT** *eventMask*)

**Parameters:**

*hLibrary*: Library handle.

*eventMask*: Pointer to library server asynchronous event mask. The following classes of events can be controllled:

- Cartridge import, export, format, and delete all file events
- File add, delete, and rename events
- Transport on-line/off-line events
- Extended data for import, export, and format on command completion

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsSetFileDescription

**#include "tekpls.h"**

**BOOL PlsSetFileDescription**(**PlsFileHandle** *hFile*,
**PlsFileDescriptionStr** *fileDescription*)

**Parameters:**

| | |
|---|---|
| *hFile*: | Filename (handle). |
| *fileDescription*: | A string to be associated with each file. A user or application can store descriptive or control data with each file using this string. |

**Return Value:** TRUE if successful; otherwise, FALSE.

this command is a synchronous command and the user need not check the status of the transaction as is in the previous case.

## PlsSetLocationString

**#include "tekpls.h"**

**BOOL PlsSetLocationString(PlsCartridgeHandle** *hCartridge*,
**PlsLocationInfoStr** *locationInfo*)

**Parameters:**

| | |
|---|---|
| *hCartridge*: | Cartridge handle. |
| *locationInfo*: | Location information string. This indicates where the cartridge is stored when it is out of library. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsSetModes

**#include "tekpls.h"**

**BOOL PlsSetModes(PlsLibraryHandle** *hLibrary*,
**PlsModes** **pModes*)

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *pModes*: | Operating modes. The following mode variables are supported, a word with single-bit modes:<br>• a bit that enables transport queuing.<br>• a bit that enables cartridge queuing. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsSetPath

**#include "tekpls.h"**

**BOOL PlsSetPath(PlsLibraryHandle** *hLibrary*,
**PlsPathStr** *path*)

Sets current directory path.

**Parameters:**

| | |
|---|---|
| *hLibrary*: | Library handle. |
| *path*: | Directory path name. |

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** This function should be used set the path before archiving/restoring a data file or clip file.

## PlsUnloadTransport

**#include "tekpls.h"**

**BOOL PlsUnloadTransport**(**PlsTransportHandle** *hTransport*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hTransport*: | Transport handle. |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## PlsUpdateCartridge

**#include "tekpls.h"**

**BOOL PlsUpdateCartridge**(**PlsTransportHandle** *hTransport*,
**PlsCartridgeHandle** *hCartridge*,
**PlsTransactionHandle** **phTransaction*)

**Parameters:**

| | |
|---|---|
| *hTransport*: | Transport handle. |
| *hCartridge*: | Cartridge handle. |
| *phTransaction*: | Pointer to transaction handle. |

**Return Value:** TRUE if successful; otherwise, FALSE.

# The Rem library

The TekRem library and associated “DLL” are used to implement the remoting of the API. The following calls are necessary to any application in a Win32 environment that wishes to use these features.

The first object to be obtained will be a *ConnectHandle*. The ConnectHandle identifies the Profile with which the application wants to communicate. If the application is running on the Profile, it can eliminate this object by using the defined value of LOCAL_CONNECTION.

## RemCloseConnection

**#include "tekrem.h"**

**BOOL RemCloseConnection**(**ConnectHandle** *connHandle*)

This function closes the connection which the caller opened to the target system. The call returns the status of the attempt.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** TRUE if the connection is closed; otherwise, FALSE.

**Comments:** All handles in the system have the connection embedded within them. To access the **ConnectHandle** (which represents the raw connection to the external node), call *RemIsolateConnection* on page 278 with the handle.

**See Also:** *RemOpenConnection* on page 280.

## RemGetConnectionName

**#include "tekrem.h"**

**BOOL RemGetConnectionName**(**ConnectHandle** *connHandle*,
**CHAR*** *nameBuffer*,
**UINT** *bufferLength*)

This function copies the node name of the connection to the given buffer.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *nameBuffer:* | The pointer to the buffer into which the connection name is copied. |
| *bufferLength:* | The length of the buffer into which the connection name is copied. The name is truncated if it exceeds the buffer length. |

**Return Value:** TRUE if the connection name is found; otherwise, FALSE.

## RemGetMajorVersion

**#include "tekrem.h"**

**UINT RemGetMajorVersion**(*void*)

This function returns the major version number of the TekRem library, the number which appears to the left of the decimal point. For example, the major number in version 1.23 is **1**.

**Parameters:** None.

**Return Value:** This function returns the major number of the TekRem library.

## RemGetMinorVersion

**#include "tekrem.h"**

**UINT RemGetMinorVersion**(*void*)

This function returns the minor version number of the TekRem library, the number which appears to the right of the decimal point. For example, the minor number in version 1.23 is **23**.

**Parameters:** None.

**Return Value:** This function returns the minor number of the TekRem library.

## RemGetNodeName

**#include "tekrem.h"**

**BOOL RemGetNodeName**(**char*** *nameBuffer*,
**UINT** *bufferLength*)

This function determines if a connection is local or remote.

**Parameters:**

*nameBuffer:* The buffer which holds the returned name of the local machine.
*bufferLength:* The length of the given buffer.

**Return Value:** TRUE if the name is returned; FALSE if the buffer is not big enough.

**Comments:** This function does not truncate the length of the buffer. The maximum allowable host name size is 20.

## RemIsHandleRemote

**#include "tekrem.h"**

**BOOL RemIsHandleRemote(ConnectHandle** *connHandle*)

This function determines if a connection is local or remote.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** TRUE if the *connHandle* is a remote handle; otherwise, FALSE.

**Comments:** All handles in the system have the connection embedded within them. To access the **ConnectHandle** (which represents the raw connection to the external node), call *RemIsolateConnection* with the handle.

## RemIsolateConnection

**#include "tekrem.h"**

**UINT RemIsolateConnection(UINT** *Handle*)

This function isolates the actual connection handle from a combined, embedded handle.

**Parameters:**

*Handle:* The other Tek library handle.

**Return Value:** This function returns connection handle portion of the combined embedded handle.

**Comments:** All handles in the system have the connection embedded within them. To access the **ConnectHandle** (which represents the raw connection to the external node), call this function with the handle.

## RemIsolateHandle

**#include "tekrem.h"**

**UINT RemIsolateHandle**(**UINT** *Handle*)

This function isolates the actual handle from a combined, embedded handle.

**Return Value:** This function returns local handle portion of the combined embedded handle.

**Comments:** This function provides access to the local portion of the handle, without the **ConnectHandle** embedded.

## RemOpenConnection

**#include "tekrem.h"**

**BOOL RemOpenConnection**(**ConnectType** *connType*,
**UINT** *connNumber*,
**const CHAR*** *hostName*,
**ConnectHandle*** *connHandle*)

This function connects the caller to the desired target system. The call returns the connection handle and the status of the attempt.

**Parameters:**

*connType:* Indicates the desired kind of connection and dictates the meaning of the other passed parameters. One of the following:

| | |
|---|---|
| 0 - ConnectLocal | Other parameters have no meaning. |
| 1 - Reserved | |
| 2 - ConnectEthernet | *hostName* is used; *connNumber* is not used. |

*connNumber:* The serial port number.

*hostName:* The node name or IP address. (TCP/IP sockets).

*connHandle:* The connection handle value returned.

**Return Value:** TRUE if the *connHandle* contains the new connection handle; otherwise, FALSE.

**Comments:** If a local connection is desired, use LOCAL_CONNECTION instead of connHandle in any call requiring it.

***NOTE: When you include tekrem.h, remtypes.h is already included, so it is not necessary to include remtypes.h separately.***

# The Vdr library

The Vdr library is responsible for sharing the embedded system's resources and for the transport control required to record and play multimedia movies.

## VdrActivateResource

**#include "tekvdr.h"**

**BOOL VdrActivateResource(ResourceHandle** *resourceHandle*)

This function activates the given resource. Playback and record will occur for resources only when the resource is active. Inactive audio resources will produce silence; the output from inactive video resources is undefined. The default state of a resource is active after receiving the handle from *VdrAllocateResource*.

**Parameters:**

*resourceHandle:* The handle to the resource returned by *VdrAllocateResource*.

**Return Value:** TRUE if the resource was activated; otherwise, FALSE.

**See Also:** *VdrAllocateResource* on page 283.

## VdrAllocateChannel

**#include "tekvdr.h"**

**VdrHandle VdrAllocateChannel(ConnectHandle** connHandle,
**char*** strChannel,
**UINT** nFlags,
**char*** strAppName,
**ChannelInfo*** pChanInfo )

This function opens a port, allocates the resources associated with the given channel name, connects those resources, sets uncompressed VBI lines (if any), and sets initial video quality.

**Parameters:**

| | |
|---|---|
| *connHandle* | The value returned from RemOpenConnection or LOCAL_CONNECTION. |
| *strChannel* | Name of a channel defined using the Configuration Manager utility. |
| *nFlags* | Currently unused. |
| *strAppName* | The name of the application making this call. Used for showing channel owner information. |
| *pChanInfo* | Pointer to a buffer that will contain channel resource use, resource handles, and error information |

**Return Value:** The port handle if successful; otherwise, NULL. When VdrAllocateChannel fails

## VdrAllocateResource

**#include "tekvdr.h"**

**ResourceHandle VdrAllocateResource(VdrHandle** *vdrHandle,*
**ResourceType** *resourceType,*
**UINT** *resourceNumber*)

This function gains control of the given physical resource. See comments below.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*resourceType:* The type of resource to control. See details below.

*resourceNumber:* Varies depending on the type of resource and how the particular disk recorder is configured.

**Return Value:** This function returns the handle to the allocated resource. If another port is already using the given resource, returns NULL.

**Comments:** When two concurrently running processes or threads attempt to use *VdrAllocateResource*, the possibility of deadly embrace exists. Here's a brief example:

Processes A and B both require use of video codec #1 and audio channel #1. Process A calls *VdrAllocateResource* and gains control of codec #1. The Windows NT operating system then suspends process A while process B runs. Process B calls *VdrAllocateResource* and gains control of audio channel #1. Process B calls *VdrAllocateResource* again and attempts to gain control of video codec #1; the call fails because process A already controls this resource. NT then suspends process B while process A runs again. Process A calls *VdrAllocateResource* again and tries to gain control of audio channel #1; the call fails because process B already controls this resource. At this point, both processes are blocked.

It is best to claim all known needed resources before using any resources. If any calls to *VdrAllocateResource* fail, all resources should be released by calls to *VdrReleaseResource*; the process should sleep for 10-100ms (random sleep times in this range are best), and then attempt to claim the resources again.

To avoid a conflict, always allocate resources in the order given in *Table 20* below. These resource types can be used in calls to *VdrAllocateResource* and *VdrGetResourceConnectionHandle*.

***NOTE: Resources should be allocated low to high.***

**Table 20. Vdr resource types**

| Resource type | Description |
|---|---|
| **...**<br>**ResourceAudioCodec** | Gains control of the given audio channel. Only one Vdr port can control a given channel at any time. During recording, the given codec can be used to record audio; during playback, to play audio. |
| Usage | (... **ResourceAudioCodec, UINT** *channelNumber*) |
| Arguments | *channelNumber*: [0... 15], number of audio channel to be controlled. |

**Table 20. Vdr resource types (Continued)**

| Resource type | Description |
|---|---|
| **...**<br>**ResourceAudioInput** | Audio input channel. The intput channels represent the back-panel outputs from the video server. Only one Vdr port can control a given input channel at any time. |
| Usage | (... **ResourceAudioOutput, UINT** *channelNumber*) |
| Arguments | *channelNumber*: [0... 15] |
| **...**<br>**ResourceAudioOutput** | Audio output channel. The output channels represent the back-panel outputs from the video server. Only one Vdr port can control a given output channel at any time. |
| Usage | (... **ResourceAudioOutput, UINT** *channelNumber*) |
| Arguments | *channelNumber*: [0... 31] |
| **...**<br>**ResourceDvcpro25Codec** | Gains control of the given DVCPRO codec. Only one Vdr port can control a given codec at any time. During recording, the given codec can be used to record video; during playback, to play video. |
| Usage | (... **ResourceDvcpro25Codec**, **UINT** *codecNumber*) |
| Arguments | *codecNumber*: Number of DVCPRO video codec to be controlled. |
| **...**<br>**ResourceDvcpro50Codec** | Gains control of the given DVCPRO codec. Only one Vdr port can control a given codec at any time. During recording, the given codec can be used to record video; during playback, to play video. |
| Usage | (... **ResourceDvcpro50Codec**, **UINT** *codecNumber*) |
| Arguments | *codecNumber*: Number of DVCPRO video codec to be controlled. |
| **...**<br>**ResourceGenTcInput** | Timecode generator (connected in the same way as a timecode input). Only one Vdr port can control a given generator at any time, though the generator can be connected to timecode recorders and timecode outputs belonging to ports other than the port controlling the generator. |
| Usage | (... **ResourceGenTcInput, UINT** *channelNumber*) |
| Arguments | *channelNumber:* Number of timecode generator. |
| **...**<br>**ResourceGpiInput** | A GPI input. The Profile XP Configuration Manager sets the GPI active state to either grounded or open (grounded by default), and defines the minimum pulse width required for an input to be considered active (2 fields by default). GPI inputs are polled every video field. GPI inputs can be shared |
| Usage | (... **ResourceGpiInput, UINT** *gpiInputResource*) |
| Arguments | *gpiInputResource.* [0... 7] Number of GPI input. |
| **...**<br>**ResourceGpiOutput** | A GPI output. The Profile XP Configuration Manager sets the GPI active state to either high or low (high by default), and defines the duration of the active state (pulse width, 60 fields by default) when the output is triggered. GPI outputs are dedicated. |
| Usage | (... **ResourceGpiOutput, UINT** *gpiOutputResource*) |
| Arguments | *gpiOutputResource.* [0... 7] Number of GPI output. |
| **...**<br>**ResourceJpegCodec** | Gains control of the JPEG given codec. Only one Vdr port can control a given codec at any time. During recording, the given codec can be used to record video; during playback, to play video. |
| Usage | (... **ResourceJpegCodec**, **UINT** *codecNumber*) |
| Arguments | *codecNumber*: [0... 3], number of JPEG codec to be controlled. |

**Table 20. Vdr resource types (Continued)**

| Resource type | Description |
|---|---|
| **...**<br>**ResourceMpegDecoder** (same as **ResourceMpegStdDefDecoder)** | Gains control of the given MPEG decoder. Only one Vdr port can control a given decoder at any time. During playback, the given decoder can be used to play video. |
| Usage | (... **ResourceMpegDecoder**, **UINT** *decoderNumber*) |
| Arguments | *decoderNumber*: Number of video decoder to be controlled. |
| **...**<br>**ResourceMpegEncoder** (same as **ResourceMpegStdDefEncoder)** | Gains control of the given MPEG encoder. Only one Vdr port can control a given encoder at any time. During recording, the given encoder can be used to record video. |
| Usage | (... **ResourceMpegEncoder**, **UINT** *encoderNumber*) |
| Arguments | *encoderNumber*: Number of video encoder to be controlled. |
| **...**<br>**ResourceMpegHiDefDecoder** | Gains control of the given High Definition MPEG decoder. Only one Vdr port can control a given decoder at any time. During playback, the given decoder can be used to play video. |
| Usage | (... **ResourceMpegHiDefDecoder**, **UINT** *decoderNumber*) |
| Arguments | *decoderNumber*: Number of video decoder to be controlled. |
| **...**<br>**ResourceMpegHiDefEncoder** | Gains control of the given High Definition MPEG encoder. Only one Vdr port can control a given encoder at any time. During recording, the given encoder can be used to record video. |
| Usage | (... **ResourceMpegHiDefEncoder**, **UINT** *encoderNumber*) |
| Arguments | *encoderNumber*: Number of video encoder to be controlled. |
| **...**<br>**ResourceMpegStdDefCodec** | Gains control of the given MPEG codec on the PVS1100 only. Only one Vdr port can control a given codec at any time. During recording, the given codec can be used to record video; during playback, to play video. |
| Usage | (... **ResourceMpegStdDefCodec**, **UINT** *codecNumber*) |
| Arguments | *codecNumber*: Number of video codec to be controlled. |
| **...**<br>**ResourceLtcInput** | Longitudinal timecode input. The input channels represent the back-panel inputs to the video server. |
| Usage | (... **ResourceLtcInput, UINT** *channelNumber*) |
| Arguments | *channelNumber*: Number of longitudinal timecode input. |
| **...**<br>**ResourceLtcOutput** | Longitudinal timecode output. The output channels represent the back-panel inputs to the video server. Only one Vdr port can control a given output channel at any time. |
| Usage | (... **ResourceLtcOutput, UINT** *channelNumber*) |
| Arguments | *channelNumber*: Number of longitudinal timecode output. |
| **...**<br>**ResourceSdtiCodec** | Gains control of the given SDTI codec on the PVS1100 only. Only one Vdr port can control a given codec at any time. During recording, the given codec can be used to record video; during playback, to play video. |
| Usage | (... **ResourceSdtiCodec**, **UINT** *codecNumber*) |
| Arguments | *codecNumber*: Number of video codec to be controlled. |
| **...**<br>**ResourceTimecodeRecorder** | Timecode recorder. The timecode recorder is used to record and playback timecode from timecode media files. Only one Vdr port can control a given recorder at any time. |
| Usage | (... **ResourceTimecodeRecorder, UINT** *channelNumber*) |
| Arguments | *channelNumber*: Number of timecode recorder. |

**Table 20. Vdr resource types (Continued)**

| Resource type | Description |
|---|---|
| **...**<br>**ResourceVideoCodec** | Used with the *CfgGetCodecAttributes* command only. Allows the application to determine the given codec's capabilities. |
| Usage | (... **ResourceVideoCodec, UINT** *codecNumber*) |
| Arguments | *codecNumber*: Number of video codec to be queried. |
| **...**<br>**ResourceVideoCombiner** | 2 input, 1 key, 1 output video mixer stage. (optional) |
| Usage | (... **ResourceVideoCombiner, UINT** *combinerNumber*) |
| Arguments | *combinerNumber*: [0 - n] (n = 2 x the number of video combiner boards. Up to two video combiner boards may be installed in a PDR system.) |
| **...**<br>**ResourceVideoInput** | Gains access to the given video input channel. The input channels represent the back-panel inputs to the video server. One input channel can be shared among many ports. |
| Usage | (... **ResourceVideoInput, UINT** *channelNumber*) |
| Arguments | *channelNumber*: [0... 31] |
| **...**<br>**ResourceVideoOutput** | Gains control of the given video output channel. The output channels represent the back-panel outputs from the video server. Only one Vdr port can control a given output channel at any time. |
| Usage | (... **ResourceVideoOutput, UINT** *channelNumber*) |
| Arguments | *channelNumber*: [0 .. 31] |
| **...**<br>**ResourceVitcInput** | Vertical interval timecode input. The input channels represent the back-panel inputs to the video server. |
| Usage | (... **ResourceVitcInput, UINT** *channelNumber*) |
| Arguments | *channelNumber*: Number of vertical interval timecode input. |
| **...**<br>**ResourceVitcOutput** | Vertical interval timecode output. The output channels represent the back-panel inputs to the video server. Only one Vdr port can control a given output channel at any time. |
| Usage | (... **ResourceVitcOutput, UINT** *channelNumber*) |
| Arguments | *channelNumber*: Number of vertical interval timecode input. |

**See Also:** *CfgGetCodecAttributes* on page 64, *VdrGetResourceConnectionHandle* on page 387, and *VdrReleaseResource* on page 409.

## VdrAttachFittedMedia

**#include "tekvdr.h"**

**MediaHandle VdrAttachFittedMedia**(**const CHAR*** *mediaFilePath,*
**ResourceHandle** *resourceHandle,*
**UINT** *duration,*
**MediaHandle** *beforeMediaHandle,*
**ShiftMode** *shiftMode*)

This function allows a piece of media, from start to finish, to be placed on the timeline for a given duration, in fields.

**Parameters:**

*mediaFilePath:* The name of the collection of media files that represent the movie.

*resourceHandle:* A **ResourceHandle** value for a codec or timecode recorder returned by *VdrAllocateResource*. Entries which are not **ResourceAudioCodec**, **ResourceJpegCodec, ResourceMpegStdDefCodec, ResourceMpegDecoder, ResourceMpegHiDefDecoder, ResourceDvcpro25Codec, ResourceDvcpro50Codec,** or **ResourceTimecodeRecorder** will be ignored.

*duration:* The length of the attached media in fields.

*beforeMediaHandle:*The handle to the media before which *movieName* should be inserted.

*shiftMode:* See *VdrAttachMedia* for details. The shift mode applies to the media files in each of the resources represented by *resourceHandles*.

**Return Value:** This function returns the handle to the attached media. If the media could not be attached, the function returns NULL.

**Comments:** The timeline duration fitting APIs allow automated control of timeline playback by “fitting” pieces of media onto the timeline for a given duration. The video subsystem will playback the timeline segments based on how many actual fields of video spanned the given duration. The formula is “Mark-out (or Mark-in) / duration”. This will occur automatically while each individual piece of media is played.

This function also allows scheduling (with new audio) of spots, in a commercial spot insertion environment, to “fit” an actual scheduled duration without having to constantly monitor and affect rate changes.

## VdrAttachFittedMediaWithMarks

**#include "tekvdr.h"**

**MediaHandle VdrAttachFittedMediaWithMarks(const CHAR*** *mediaFilePath,*
**ResourceHandle** *resourceHandle,*
**UINT** *duration,*
**MediaHandle** *beforeMediaHandle,*
**ShiftMode** *shiftMode,*
**INT** *markIn,*
**INT** *markOut*)

This function allows a piece of media--from the specified in-point to the specified out-point--to be placed on the timeline for a given duration, in fields.

**Parameters:**

*mediaFilePath:* The name of the collection of media files that represent the movie.

*resourceHandle:* A **ResourceHandle** value for a codec or timecode recorder returned by *VdrAllocateResource*. Entries which are not **ResourceAudioCodec**, **ResourceJpegCodec, ResourceMpegStdDefCodec, ResourceMpegDecoder, ResourceMpegHiDefDecoder, ResourceDvcpro25Codec, ResourceDvcpro50Codec,** or **ResourceTimecodeRecorder** will be ignored.

*duration:* The length of the attached media in fields.

*beforeMediaHandle:* The handle to the media before which *movieName* should be inserted.

*shiftMode:* See *VdrAttachMedia* for details. The shift mode applies to the media files in each of the resources represented by *resourceHandles.*

*markIn:* A pointer to **UINT** into which the *MarkIn* value can be stored.

*markOut:* A pointer to **UINT** into which the *MarkOut* value can be stored.

**Return Value:** This function returns the handle to the attached media. If the media could not be attached, the function returns NULL.

**Comments:** The timeline duration fitting APIs allow automated control of timeline playback by “fitting” pieces of media onto the timeline for a given duration. The video subsystem will playback the timeline segments based on how many actual fields of video spanned the given duration. The formula is “Mark-out (or Mark-in) / duration”. This will occur automatically while each individual piece of media is played.

This function also allows scheduling (with new audio) of spots, in a commercial spot insertion environment, to “fit” an actual scheduled duration without having to constantly monitor and affect rate changes.

## VdrAttachMedia

**#include "tekvdr.h"**

**MediaHandle VdrAttachMedia(const CHAR*** *mediaFilePath,*
**ResourceHandle** *resourceHandle,*
**MediaHandle** *beforeMediaHandle,*
**ShiftMode** *shiftMode*)

This function attaches the given media file to the given resources for playback or record. If *beforeMediaHandle* is not NULL, the media file is inserted in the playback or record lists before the file represented by *beforeMediaHandle*. If *beforeMediaHandle* value is NULL, the media file is attached at the ends of the playback or record lists.

**Parameters:**

*mediaFilePath:* The path name of the media file, or NULL (see Comments).

*resourceHandle:* A Resource Handle value for a codec or timecode recorder returned by *VdrAllocateResource*.

*beforeMediaHandle:* The handle to the media before which *mediaName* should be inserted.

*shiftMode:* One of the following:

- ShiftBefore
- ShiftAfter

If *shiftMode* is **ShiftBefore**, the reference field numbers for fields in media files attached earlier than *beforeMediaHandle* will have a number of fields subtracted equal to the length (in fields) of the attached media file. If *shiftMode* is **ShiftAfter**, the reference field numbers for fields in *beforeMediaHandle* and all media files attached later in the list than *beforeMediaHandle* will have this number of fields added.

**Return Value:** This function returns the handle to the attached media file. If the file could not be attached, it returns NULL.

**Comments:** If *mediaFilePath* is NULL or the pathname's first character is OxOO, a file of BLACK media--black on video channels, silence on audio channels--will be attached; it is not possible to record into a file of BLACK media. If *mediaFilePath* did not exist before this call, a media file will be created.

## VdrAttachMediaWithMarks

**#include "tekvdr.h"**

**MediaHandle VdrAttachMediaWithMarks**(**const CHAR*** *mediaFilePath,*
**ResourceHandle** *resourceHandle,*
**MediaHandle** *beforeMediaHandle,*
**ShiftMode** *shiftMode,*
**INT** *markIn,*
**INT** *markOut*)

This function attaches the given media file to the given resource for playback or record and sets the given in-point and out-point. If *beforeMediaHandle* is not NULL, the file is inserted in the playback or record lists before the file represented by *beforeMediaHandle*; if *beforeMediaHandle* value is NULL, the file is attached at the end of the playback or record lists.

**Parameters:**

*mediaFilePath:* The path name of the media file, or NULL (see Comments).

*resourceHandle:* A handle to a codec or timecode recorder returned by *VdrAllocateResource*.

*beforeMediaHandle:* The handle to the media before which *mediaName* should be inserted.

*shiftMode:* One of the following:

- ShiftBefore
- ShiftAfter

If *shiftMode* is **ShiftBefore**, the reference field numbers for fields in media files attached earlier in the list than *beforeMediaHandle* will have a number of fields subtracted equal to the length (in fields) of the attached media file. If *shiftMode* is **ShiftAfter**, the reference field numbers for fields in *beforeMediaHandle* and all media files attached later in the list than *beforeMediaHandle* will have this number of fields added.

*markIn* The desired in-point.

*markOut* The desired out-point.

**Return Value:** This function returns the handle to the attached media file. If the file could not be attached, it returns NULL. The function also sets the in-point and out-point to the given values and will return NULL if:

*markIn* < 0,
*markOut* < *0*, or
*markIn* > *markOut*

**Comments:** If *mediaFilePath* is NULL or the pathname's first character is OxOO, a file of BLACK media--black on video channels, silence on audio channels--will be attached; it is not possible to record into a file of BLACK media. If *mediaFilePath* did not exist before this call, a media file will be created.

## VdrAttachMovie

**#include "tekvdr.h"**

**MovieHandle VdrAttachMovie(const CHAR*** *movieName*,
**UINT** *numResources*,
**ResourceHandle*** *resourceHandles*,
**MovieHandle** *beforeMovieHandle*,
**ShiftMode** *shiftMode*,
**MarkMode** *markMode*)

This function attaches the given movie (collection of media files) to the given resources for playback or record. If *beforeMovieHandle* is not NULL, the media files are inserted in the playback or record lists before the media files represented by *beforeMovieHandle*. If *beforeMovieHandle* value is NULL, the media files are attached at the ends of the playback or record lists. If *movieNam*e is NULL, attaches Null media files composed of video black for video channels, and audio silence for audio channels (no recording can occur for the movie handle returned when a null media file is attached). If *movieName* did not exist before this call, a movie will be created with a collection of media files, one per resource. Attaching a movie automatically resets the minimum and maximum play/record positions. (See *VdrSetMinPosition* and *VdrSetMaxPosition*.)

You can attach an MPEG movie to an MPEG encoder only if the movie consists entirely of I-frames. Attempting to attach a long-GOP MPEG movie to an MPEG encoder will fail.

If media attachment fails because a media file is corrupt, all other media files for the movie are detached and a NULL handle is returned.

**Parameters:**

*movieName:* The complexMovieName that represents the movie.

*numResources:* Number of resources in the array *resourceHandles*.

*resourceHandles:* A pointer to an array of **ResourceHandle**. Entries which are not **ResourceAudioCodec**, **ResourceJpegCodec, ResourceMpegStdDefCodec, ResourceMpegEncoder, ResourceMpegDecoder, ResourceMpegHiDefDecoder, ResourceMpegHiDefDecoder, ResourceDvcpro25Codec, ResourceDvcpro50Codec,** or **ResourceTimecodeRecorder** will be ignored.

*beforeMovieHandle:*The handle to the movie before which *movieName* should be inserted.

*shiftMode:* See *VdrAttachMedia* for details. The shift mode applies to the media files in each of the resources represented by *resourceHandles*.

*markMode:* Specifies how the movie will be treated if all media files are not of the same duration.

**Return Value:** This function returns the handle to the attached movie. If the movie could not be attached, returns NULL.

**Comments:** If the movie exists in the inventory, the function will try to open the movie Exclusive so that the addition of tracks and media segments to match the number of codec and recorder resources will not cause the movie description

to exceed its current size. If the PdrOpenExclusive fails, the function will open the movie shared.

For each codec or recorder resource for which there is no track in the movie description, a track with a single new media file will be created.

After the movie is attached, the movie is closed and re-opened in the shared mode. The library will keep a MovieToken for the movie as long as the movie is attached to the port timeline.

**See Also:** *VdrAttachMedia* on page 289, *VdrSetMaxPosition* on page 434, and *VdrSetMinPosition* on page 442.

## VdrAttachMovieWithMarks

**#include "tekvdr.h"**

**MovieHandle VdrAttachMovieWithMarks(const CHAR*** *movieName*,
**UINT** *numResources*,
**ResourceHandle*** *resourceHandles*,
**MovieHandle** *beforeMovieHandle*,
**ShiftMode** *shiftMode*,
**MarkMode** *markMode*,
**INT** *markIn*,
**INT** *markOut*)

This function attaches the given movie (collection of media files) to the given resources for playback or record and sets the given in-point and out-point. If *beforeMovieHandle* is not NULL, the media files are inserted in the playback or record lists before the media files represented by *beforeMovieHandle*. If *beforeMovieHandle* value is NULL, the media files are attached at the ends of the playback or record lists. If *movieName* is NULL, the function attaches Null media files composed of video black for video channels, and audio silence for audio channels (no recording can occur for the movie handle returned when a null media file is attached). If *movieName* did not exist before this call, a collection of media files will be created, one per resource. Attaching a movie automatically resets the minimum and maximum play/record positions. (See *VdrSetMaxPosition* and *VdrSetMinPosition*.)

If media attachment fails because a media file is corrupt, all other media files for the movie are detached and a NULL handle is returned.

**Parameters:**

*movieName:* The complex Movie Name of the collection of media files that represent the movie.

*numResources:* Number of resources in the array *resourceHandles*.

*resourceHandles:* A pointer to an array of **ResourceHandle**. Entries which are not **ResourceAudioCodec**, **ResourceJpegCodec, ResourceMpegStdDefCodec, ResourceMpegEncoder, ResourceMpegDecoder, ResourceMpegHiDefDecoder, ResourceMpegHiDefDecoder, ResourceDvcpro25Codec, ResourceDvcpro50Codec,** or **ResourceTimecodeRecorder** will be ignored.

*beforeMovieHandle:* The handle to the movie before which *movieName* should be inserted.

*shiftMode:* See *VdrAttachMedia* for details. The shift mode applies to the media files in each of the resources represented by *resourceHandles*.

*markMode:* Specifies how the movie will be treated if all media files are not of the same duration.

*markIn:* The desired in-point.

*markOut:* The desired out-point.

**Return Value:** This function returns the handle to the attached movie. If the movie could not be attached, returns NULL.

**Comments:** If the movie exists in the inventory, the function will try to open the movie Exclusive so that the addition of tracks and media segments to match the number of codec and recorder resources will not cause the movie description

to exceed its current size. If the PdrOpenExclusive fails, the function will open the movie shared.

For each codec or recorder resource for which there is no track in the movie description, a track with a single new media file will be created.

After the movie is attached, the movie is closed and re-opened in the shared mode. The library will keep a MovieToken for the movie as long as the movie is attached to the Port Timeline.

**See Also:** *VdrAttachMedia* on page 289, *VdrSetMaxPosition* on page 434, and *VdrSetMinPosition* on page 442.

## VdrAttachOpenMovie

**#include "tekvdr.h"**

**MovieHandle VdrAttachOpenMovie(MovieToken** *movieToken,*
**UINT** *numResources,*
**ResourceHandle*** *resourceHandles,*
**MovieHandle** *beforeMovieHandle,*
**ShiftMode** *shiftMode,*
**MarkMode** *markMode*)

This function attaches the given movie to the given resources for playback or record. If *beforeMovieHandle* is not NULL, the media files are inserted in the playback or record lists before the media files represented by *beforeMovieHandle*. If *beforeMovieHandle* is NULL, the media files are attached at the end of the playback or record lists. If the function fails, no media files will be attached and a NULL handle is returned.

**Parameters:**

*movieToken:* The movie token that represents an open movie from the inventory.

*numResources:* Number of resources in the array *resourceHandles*.

*resourceHandles:* A pointer to an array of **ResourceHandle**. Entries which are not **ResourceAudioCodec**, **ResourceJpegCodec, ResourceMpegStdDefCodec, ResourceMpegStdDefCodec, ResourceMpegDecoder, ResourceMpegHiDefDecoder, ResourceMpegHiDefDecoder, ResourceDvcpro25Codec, ResourceDvcpro50Codec,** or **ResourceTimecodeRecorder** will be ignored.

*beforeMovieHandle:* The handle to the movie before which *movie* should be inserted.

*shiftMode:* See *VdrAttachMedia* for details. The shift mode applies to the media files in each of the resources represented by *resourceHandles*.

*markMode:* Specifies how the movie will be treated if all media files are not of the same duration.

**Return Value:** This function returns the handle to the attached movie. If the movie could not be attached, returns NULL.

**Comments:** The movie will have additional tracks and media files added to its description if the number of codec and recorder resources in the array is greater than the corresponding number of tracks in the movie. The movie should not be closed before it is detached (VdrDetachMovie).

## VdrAttachOpenMovieWithMarks

**#include "tekvdr.h"**

**MovieHandle VdrAttachOpenMovieWithMarks(MovieToken** *movieToken,*
**UINT** *numResources,*
**ResourceHandle*** *resourceHandles,*
**MovieHandle** *beforeMovieHandle,*
**ShiftMode** *shiftMode,*
**MarkMode** *markMode,*
**INT** *markIn,*
**INT** *markOut*)

This function attaches the given movie to the given resources for playback or record. If *beforeMovieHandle* is not NULL, the media files are inserted in the playback or record list before the media files represented by *beforeMovieHandle*. If *beforeMovieHandle* is NULL, the media files are attached at the end of the list. Once attached, in-point and out-point values are set on each resource for the movie. If the function fails, no media is attached and a NULL handle is returned.

**Parameters:**

*movieToken:* The movie token that represents an open movie from the inventory.

*numResources:* Number of resources in the array *resourceHandles*.

*resourceHandles:* A pointer to an array of **ResourceHandle**. Entries which are not **ResourceAudio**, **ResourceCodec,** or **ResourceTimecodeRecorder** will be ignored.

*beforeMovieHandle:* The handle to the movie before which *movie* should be inserted.

*shiftMode:* See *VdrAttachMedia* for details. The shift mode applies to the media files in each of the resources represented by *resourceHandles.*

*markMode:* Specifies how the movie will be treated if all media files are not of the same duration.

*markIn:* The desired in-point.

*markOut:* The desired out-point.

**Return Value:** This function returns the handle to the attached movie. If the movie could not be attached, returns NULL.

**Comments:** The movie will have additional tracks and media files added to its description if the number of codec and recorder resources in the array is greater than the corresponding number of tracks in the movie. The movie should not be closed before it is detached (VdrDetachMovie).

## VdrCancelAllPendingGpiTriggerCommands

**#include "tekvdr.h"**

**BOOL VdrCancelAllPendingGpiTriggerCommands(VdrHandle** vdrHandle);

This function cancels all pending GPI trigger commands.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** TRUE if the GPI trigger commands were canceled; otherwise FALSE.

**Comments:** GPI ports assigned to NetCentral are not affected by this command.

## VdrCancelEvent

**#include "tekvdr.h"**

**BOOL VdrCancelEvent(EventHandle** *eventHandle*)

This function cancels an event.

**Parameters:**

*eventHandle:* The handle to the event returned by *VdrScheduleEvent*; *eventHandle* is invalid after this call.

**Return Value:** TRUE if the event was canceled; otherwise, FALSE.

## VdrCancelPendingTimecodeTriggerCommands

**#include "tekvdr.h"**

**BOOL VdrCancelPendingTimecodeTriggerCommands** (**VdrHandle** *vdrHandle*)

This function cancels all outstanding AtGenTc controls that have been set on the port by the following commands: *VdrDefaultEventAtGenTc*, *VdrShuttleAtGenTc*, *VdrStateEventAtGenTc*, *VdrTriggerGpiOutputAtGenTc*.

**Parameters:**

*vdrHandle*: The handle returned by *VdrOpenPortConnection*.

**Return Value:** TRUE if successful; otherwise, FALSE;

**Comments:** The port must not be in an Idle state when this function is called.

## VdrClosePort

**#include "tekvdr.h"**

**BOOL VdrClosePort(VdrHandle** *vdrHandle*)

This function closes the given embedded disk recorder system control port. The *vdrHandle* handle is not valid for use with other commands after this call. This frees the port to be used in subsequent *VdrOpenPortConnection* calls.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** TRUE if the port was closed; otherwise, FALSE.

**Comments:** A call to *VdrClosePort* will release all resource handles which have been obtained via *VdrAllocateResource* for that port. A call to *VdrReleaseResource* will detach all media handles obtained via *VdrAttachMedia* for the resource. The resource and media handles cannot be used after these calls.

## VdrCuePlay

**#include "tekvdr.h"**

**BOOL VdrCuePlay(VdrHandle** *vdrHandle*,
**double** *expectedRate*)

This function cues up for playback all audio, video, or timecode media files that have been attached to channels by *VdrAttachMedia* calls from this port.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |
| *expectedRate:* | If *VdrShuttle* is to be used to enter motion play, it should be the same as the initial value for *shuttleRate*. If *VdrJog* is to be used to enter still play, it should be the (*numberOfFields* * 1.0), where *numberOfFields* is the expected step size for *VdrJog*. Mismatches in the parameters for *VdrCuePlay* and these commands are allowed, but will produce less-than-optimal playback on startup of motion or still play. |

**Return Value:** TRUE if cue-up occurred; otherwise, FALSE.

## VdrCueRecord

**#include "tekvdr.h"**

**BOOL VdrCueRecord(VdrHandle** *vdrHandle*)

This function cues up for playback all audio, video, or timecode media files that have been attached to channels by *VdrAttachMedia* calls from this port.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** TRUE if cue-up occurred; otherwise, FALSE.

## VdrDeactivateResource

**#include "tekvdr.h"**

**BOOL VdrDeactivateResource**(**ResourceHandle** *resourceHandle*)

This function deactivates the given resource. See comments under *VdrActivateResource*.

**Parameters:**

*resourceHandle:* The handle to the resource returned by *VdrAllocateResource*.

**Return Value:** TRUE if the resource was deactivated; otherwise, FALSE.

**See Also:** *VdrActivateResource* on page 281.

## VdrDefaultEvent

**#include "tekvdr.h"**

**BOOL VdrDefaultEvent(VdrHandle** *vdrHandle*,
**Handle** NULL,
**EventType** *type*, ...)

This function specifies an event which takes effect immediately but can be overridden by *VdrStateEvent* and *VdrScheduleEvent* commands. Normally, VdrDefaultEvent is considered to be active during idle, record-cue, and record states of the port; it can be active during play and play-cue states if it has not been overridden by other events.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |
| *NULL:* | Required parameter. |
| *type:* | Specific event type and supporting parameters. (See *Table 21* below). |

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** VdrDefaultEvent is equivalent to VdrStateEvent with the state mask set to EventStateAll; other event commands may be issued after the Default Event, which will override the states at which the Default Event is active.

The types of events and their parameters are listed in *Table 21* below.

**See Also:** *VdrScheduleEvent* on page 413,
*VdrStateEvent* on page 462.

**Table 21. Vdr events**

| Event type | Description |
|---|---|
| **EventConnectResources** | Connects the source resource to the destination resource. |
| Usage | (... **EventConnectResources**, **ResourceHandle** *sourceHandle*, **ResourceHandle** *destinationHandle*) |
| Arguments | *sourceHandle*: handle to a video or timecode resource providing output, returned by **VdrAllocateResource** or **VdrGetResourceConnectionHandle**.<br>*destinationHandle*: handle to a video or timecode resource requiring input, returned by **VdrAllocateResource**. |
| **EventMixAudio** | Mixes the output from the given audio codec to the given level at the given audio output, changing the mix over the given number of fields. (Compare with definition in *Table 23* on page 414.) |
| Usage | (... **EventMixAudio**, **ResourceHandle** *audioCodecHandle*, **ResourceHandle** *audioOutputHandle*, **double** *mixLevel*, **UINT** *numFields*) |
| Arguments | *audioCodecHandle*: handle to the audio codec, returned by **VdrAllocateResource** or **VdrGetResourceConnectionHandle**.<br>*audioOutputHandle*: handle to the audio output, returned by **VdrAllocateResource**.<br>*mixLevel:* [.0 .. 1.0]; level at which the output of the audio codec should be mixed into the audio output. 1.0 represents an unattentuated mix.<br>*numFields*: duration (in video fields) of the change in mix from the current mix to the given mix. |
| **EventSetGtcTime** | Places the specified timecode pattern into the specified timecode generator. |
| Usage | (... **EventSetGtcTime**, **ResourceHandle** *timecodeGeneratorHandle*, **INT** *pattern*, **INT** *fieldNumber*) |
| Arguments | *timeCodeGeneratorHandle*: handle to timecode generator, returned by **VdrAllocateResource**.<br>*pattern*: timecode value encoded as four binary numbers, each a single-byte wide.<br>*fieldNumber*: the field number at which to place the *pattern*.<br>**UINT** pattern = (hours << 24) \| (minutes << 16) \| (seconds << 8) \| (fields) |
| **EventSetGtcBits** | Puts the specified user bit pattern into the timecode generator. |
| Usage | (... **EventSetGtcBits**, **ResourceHandle** *timeCodeGeneratorHandle*, **UINT** *bitpattern*) |
| Arguments | *timeCodeGeneratorHandle*: handle to timecode generator, returned by **VdrAllocateResource**.<br>*bitpattern*: As defined in proposed revised standard SMPTE 12M, April 1993. |

**Table 21. Vdr events**

| Event type | Description |
|---|---|
| **EventVoiceOver** | Mixes the audio input at the given level to the audio output, changing the mix over the given number of fields. (Compare with definition in *Table 23* on page 414.) |
| Usage | (... **EventVoiceOver**, **ResourceHandle** *audioInputHandle*, **ResourceHandle** *audioOutputHandle*, **double** *mixLevel*, **UINT** *numFields*)<br>This event is active when the port is in CuePlay or Play state. This event is only available with the ASPB audio board. |
| Arguments | *audioInputHandle*: handle to the audio input, returned by **VdrAllocateResource** or **VdrGetResourceConnectionHandle**.<br>*audioOutputHandle*: handle to the audio output, returned by **VdrAllocateResource**.<br>*mixLevel:* [.0 .. 1.0]; level at which the audio input should be mixed into the audio output. 1.0 represents an unattentuated mix.<br>*numFields*: duration (in video fields) of the change in mix from the current mix to the given mix. |

## VdrDefaultEventAtGenTc

**#include "tekvdr.h"**

**BOOL VdrDefaultEventAtGenTc(VdrHandle** *vdrHandle*,
**Handle** *NULL*,
**ResourceHandle** *genTcHandle*,
**UINT** *time*,
**EventType** *type*, ...)

This function specifies an event which can be scheduled to take effect at a specific time. The event, like a VdrDefaultEvent, takes effect in all states in which there is not an overriding event.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |
| *NULL:* | Required parameter. |
| *genTcHandle:* | The handle of a Time Code Generator that was returned by VdrAllocateResource. The generator is the source of the time used for scheduling the event. The generator may be synchronized to any of several different timecode and position sources. |
| *time:* | The time, encoded as timecode, at which the event is to be scheduled. (See the SET_TIMECODE macro in *vdrtypes.h*) |
| *type:* | Specific event type and supporting parameters (see list below). |

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** The types of events listed in *Table 21, Vdr events* on page 305 can be used.

The time parameter must be one field ahead of the desired start time in order to avoid drop-frame issues and to maintain compatibility with the mechanism used by *VdrShuttle*.

For example, if the desired start time is 1:15:00:00, the specified time parameter must be 1:14:59:59 (the last two digits specify the field number). If drop-frame timecode is in use, and the desired start time is 10:00:00:04, the specified time parameter must be 9:59:59:59, taking into account the four dropped timecode fields.

This command must be issued no later than the latency value specified by *VdrGetStateChangeLatency* plus one field before the desired start time.

For example, if the latency value is 14 fields and the desired start time is 1:15:00:00, the command must be issued no later than 1:14:59:45. If drop-frame timecode is in use, the latency value is 14 fields, and the desired start time is 10:00:00:04, the command must be issued no later than 9:59:59:45, taking into account the four dropped timecode fields.

**See Also:** *VdrSetDefaultTcFormat* on page 421,
*VdrSetGenTcRegenSource* on page 428,
*VdrSetGenTcFormat* on page 426,
*VdrSetGenTcMode* on page 427.

## VdrDefaultEventOnGpiInputTrigger

**BOOL VdrDefaultEventOnGpiInputTrigger(VdrHandle** vdrHandle,
**HANDLE** reserved,
**ResourceHandle** gpiInputResource,
**EventType** eventType, ...);

This function specifies an event which can be scheduled to take effect upon receipt of a GPI input trigger. The event, like a VdrDefaultEvent, takes effect in all states in which there is not an overriding event.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*reserved:* Set to NULL.

*gpiInputResource:* The handle to the resource of type **ResourceGpiOutput** returned by *VdrAllocateResource*.

*eventType:* Specific event type and supporting parameters (see Comments below).

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** The types of events listed in *Table 21, Vdr events* on page 305 can be used.

## VdrDetachMedia

**#include "tekvdr.h"**

**BOOL VdrDetachMedia**(**MediaHandle** *mediaHandle*,
**ShiftMode** *shiftMode*)

This function detaches the given file media handle from its channel. The associated file will be removed from the playback or record list for the resource to which it was attached. The file will still exist in the file system. The media handle cannot be used after this call.

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

*shiftMode:* One of the following:

- ShiftBefore
- ShiftAfter

If *shiftMode* is **ShiftBefore**, the reference field numbers for fields in media files attached earlier in the list than *mediaHandle* will have a number of fields added equal to the length (in fields) of the detached media file. If *shiftMode* is **ShiftAfter**, the reference field numbers for fields in all media files attached later in the list than *mediaHandle* will have this number of fields subtracted.

**Return Value:** TRUE if the handle was detached; otherwise, FALSE.

## VdrDetachMovie

**#include "tekvdr.h"**

**BOOL VdrDetachMovie**(**MovieHandle** *movieHandle*,
**ShiftMode** *shiftMode*)

This function detaches the media handles represented by *movieHandle* from their channels. The associated files will be removed from the playback or record lists for the resources to which they were attached and the files will still exist in the embedded disk recorder file system. The movie handle cannot be used after this call. Detaching a movie automatically resets the minimum and maximum play/record positions. (See *VdrSetMaxPosition* and *VdrSetMinPosition*.)

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

*shiftMode:* See *VdrAttachMedia* for details. The shift mode applies to the media files in each of the resources represented in the movie.

**Return Value:** TRUE if the movie was detached; otherwise, FALSE.

**See Also:** *VdrAttachMedia* on page 289,
*VdrAttachMovie* on page 291,
*VdrAttachMovieWithMarks* on page 293,
*VdrSetMaxPosition* on page 434, and
*VdrSetMinPosition* on page 442.

## VdrDisableMoviePositionLimiting

**#include "tekvdr.h"**

**BOOL VdrDisableMoviePositionLimiting(VdrHandle** *vdrHandle*)

This function disables the affect of a previous call to *VdrEnableMoviePositionLimiting* for the given handle.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** TRUE if the handle was disabled; otherwise, FALSE.

**See Also:** *VdrEnableMoviePositionLimiting* on page 312, *VdrIsMoviePositionLimitingEnabled* on page 403.

## VdrEnableMoviePositionLimiting

**#include "tekvdr.h"**

**BOOL VdrEnableMoviePositionLimiting(VdrHandle** *vdrHandle*)

This function sets position limits when the following functions are called for resources allocated to the given vdrHandle:

- VdrAttachMovie
- VdrDetachMovie
- VdrSetMovieMarkIn
- VdrSetMovieMarkOut

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** TRUE if the handle was enabled; otherwise, FALSE.

**See Also:** *VdrDisableMoviePositionLimiting* on page 311,
*VdrIsMoviePositionLimitingEnabled* on page 403.

## VdrExpandMovie

**#include "tekvdr.h"**

**BOOL VdrExpandMovie**(**MovieHandle** *movieHandle*,
**UINT** *numResources*,
**ResourceHandle*** *resourceHandles*,
**MovieHandle** *beforeMovieHandle*,
**ShiftMode** *shiftMode*)

This function expands the already-attached movie to fill the given resources. This is to be used when new resources are allocated while a movie is already attached, and additional media files are to be attached to the new resources.

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

*numResources:* The number of resources in the array *resourceHandles*.

*resourceHandles:* The pointer to the new array of resource handles. See *VdrAttachMedia* for details.

*beforeMovieHandle:* See *VdrAttachMedia* for details.

*shiftMode:* See *VdrAttachMedia* for details.

**Return Value:** TRUE if successful; otherwise, FALSE.

## VdrGetAbsMaxChrQ

**#include "tekvdr.h"**

**double VdrGetAbsMaxChrQ(VdrHandle** *vdrHandle*)

This function returns the maximum chrominance quantization level for the given JPEG codec resource, which can be used in calls to *VdrSetMaxChrQ* or *VdrSetMinChrQ*.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the maximum limit for minimum or maximum chrominance Q.

**Comments:** This applies to JPEG only; it is not applicable to MPEG.

## VdrGetAbsMaxLumQ

**#include "tekvdr.h"**

**double VdrGetMaxLumQ(ResourceHandle** *resourceHandle*)

This function returns the maximum luminance quantization level for the given JPEG codec resource, which can be used in calls to *VdrSetMaxLumQ* or *VdrSetMinLumQ*.

**Parameters:**

*resourceHandle:* The handle to a resource of type ResourceJpegCodec returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the maximum limit for minimum or maximum luminance Q.

**Comments:** This applies to JPEG only; it is not applicable to MPEG.

## VdrGetAbsMinChrQ

**#include "tekvdr.h"**

**double VdrGetAbsMinChrQ(VdrHandle** *vdrHandle*)

This function returns the minimum chrominance quantization level for the given JPEG codec resource, which can be used in calls to *VdrSetMaxChrQ* or *VdrSetMinChrQ*.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the minimum limit for minimum or maximum chrominance Q.

**Comments:** This applies to JPEG only; it is not applicable to MPEG.

## VdrGetAbsMinLumQ

**#include "tekvdr.h"**

**double VdrGetAbsMinLumQ(VdrHandle** *vdrHandle*)

This function returns the minimum luminance quantization level for the given JPEG codec resource, which can be used in calls to *VdrSetMaxLumQ* or *VdrSetMinLumQ*.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the minimum limit for minimum or maximum luminance Q.

## VdrGetAudioInputLevel

**#include "tekvdr.h"**

**double VdrGetAudioInputLevel**(**ResourceHandle** *resourceHandle*)

This function returns the current setting of the audio input level (same value as parameter in successful call to *VdrSetAudioInputLevel*).

**Parameters:**

*resourceHandle:* The handle to an audio codec (**ResourceAudioCodec**).

**Return Value:** This function returns [0.0 ... 1.0] audio level. A return value of `0.0` indicates full attenuation; `1.0` indicates no attenuation.

## VdrGetAudioLevel

**#include "tekvdr.h"**

**UINT VdrGetAudioLevel(ResourceHandle** *resourceHandle*)

This function returns the highest audio sample level from the current field for the given audio output.

**Parameters:**

*resourceHandle:* The handle to an audio output (**ResourceAudioOutput**).

**Return Value:** This function returns [0 ... 7ff]. A return value of `0` corresponds to minimum volume; maximum volume is board-dependent.

## VdrGetAudioOutputLevel

**#include "tekvdr.h"**

**double VdrGetAudioOutputLevel(ResourceHandle** *resourceHandle*)

This function returns the current setting of the audio output level (same value as parameter in successful call to *VdrSetAudioOutputLevel*).

**Parameters:**

*resourceHandle:* The handle to an audio output (**ResourceAudioOutput**).

**Return Value:** This function returns [0.0 ... 1.0] audio level. A return value of `0.0` indicates full attenuation; `1.0` indicates no attenuation.

## VdrGetAudioWindow

**#include "tekvdr.h"**

**UINT VdrGetAudioWindow**(**ResourceHandle** *resourceHandle*)

This function returns the repetition interval (in fields) for the given audio resource when audio playback occurs during still play mode.

**Parameters:**

*resourceHandle:* The handle to the audio codec returned by *VdrAllocateResource*.

**Return Value:** This function returns the repetition interval in fields.

## VdrGetBitRate

**#include "tekvdr.h"**

**double VdrGetBitRate**(**ResourceHandle** *resourceHandle*)

Return the current bit-rate used for encoding.

**Parameters:**

*resourceHandle:* The handle to the JPEG codec, MPEG, or MPEG HD encoder returned by *VdrAllocateResource*.

**Return Value:** Current bit-rate for encoding, expressed in bits/second. For example, `bitRate=36.0e6` causes encoding at 36Mbps.

## VdrGetChannelInfoList

**#include "tekvdr.h"**

**BOOL VdrGetChannelInfoList (ConnectHandle** connHandle,
**ChannelInfo*** pChanInfoList,
**UINT** bufferSize,
**int** nRequested,
**int*** pCount)

This function retrieves the channel definition identifiers.

**Parameters:**

| | |
|---|---|
| *connHandle* | The value returned from RemOpenConnection or LOCAL_CONNECTION. |
| *channelIDs* | A pointer to a buffer to receive a list of ChannelInfo structures. |
| *bufferSize* | Number of bytes in the ChanInfoList buffer. |
| *nRequested* | The number of channel identifiers to retrieve. This number can be determined by calling *VdrGetNumChannelDefs* prior to calling this function. |
| *pCount* | An out parameter for the number of channel identifiers returned. |

**Return value:** If the number of channel definitions exceeds nRequested, the excess channel identifiers will not be returned. The return value is FALSE if an error occurs while accessing the channel definitions; otherwise, TRUE.

## VdrGetCurrentFieldSize

**#include "tekvdr.h"**

**UINT VdrGetCurrentFieldSize**(**ResourceHandle** *resourceHandle*)

This function returns the compressed size of the current field for the given video codec.

**Parameters:**

*resourceHandle:* The handle to the JPEG codec, MPEG, or MPEG HD encoder returned by *VdrAllocateResource*.

**Return Value:** This function returns the compressed size (in bytes) of the current video field.

## VdrGetCurrentLumQFactor

**#include "tekvdr.h"**

**double VdrGetCurrentLumQFactor**(**ResourceHandle** *resourceHandle*)

This function returns the luminance Q factor for the current field from the given video codec.

**Parameters:**

*resourceHandle:* The handle to the JPEG codec.

**Return Value:** This function returns the luminance Q.

## VdrGetCurrentPictureStatus

**#include "tekvdr.h"**

**BOOL VdrGetCurrentPictureStatus**(**ResourceHandle** *resourceHandle*,
**UINT*** *pixSize*,
**PixCoding*** *pixCoding*,
**MpegPixStructure*** *pixStructure*)

Returns the size, coding, and structure for the picture at the current position within the given resource handle.

**Parameters:**

| | |
|---|---|
| *resourceHandle:* | The handle returned by *VdrAllocateResource*. |
| *pixSize***:** | Pointer to a variable in which the size (in bytes) of the current picture will be written. |
| *pixCoding:* | Pointer to a variable in which the coding type of the current picture will be written. |
| *pixStructure:* | Pointer to a variable in which the structure of the current picture will be written. |

**Return Value:** TRUE if the call succeeds; FALSE otherwise.

**Comments:** Any pointer value of *pixSize*, *pixCoding* or *pixStructure* which is NULL will have no value returned. For ResourceJpegCodec, the value returned for *pixCoding* will be `PixCodingI`, and the value returned for *pixStructure* will be `PixStructureField`.

## VdrGetCurrentRate

**#include "tekvdr.h"**

**double VdrGetCurrentRate(VdrHandle** *vdrHandle*)

This function returns the current play or record rate for the given Vdr port.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the current play or record rate for the given Vdr port.

## VdrGetCurrentState

**#include "tekvdr.h"**

**VdrState VdrGetCurrentState(VdrHandle** *vdrHandle*)

This function returns the current state of the given port.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** If the *vdrHandle* is not valid or if the embedded disk recorder subsystem cannot map a status pointer for the port associated with *vdrHandle*, this function returns one of the following states:

- VdrStateIdle
- VdrStateCuedPlay
- VdrStateCuedRecord
- VdrStateJogPlay
- VdrStateJogRecord
- VdrStateShuttlePlay
- VdrStateShuttleRecord
- VdrStateGoPlay
- VdrStateGoRecord
- VdrStateInvalid

## VdrGetCurrentTimeCode

**#include "tekvdr.h"**

**UINT VdrGetCurrentTimeCode**(**ResourceHandle** *resourceHandle*)

This function returns the current timecode from the given timecode resource.

**Parameters:**

*resourceHandle:* The handle to one of the following:

- ResourceLtcInput
- ResourceVitcInput
- ResourceGenTcInput
- ResourceTimecodeRecorder

**Return Value:** This function returns a 32-bit integer:

- bits 0-7: fields
- bits 8-15: seconds
- bits 16-23: minutes
- bits 24-28: hours
- bits 29-31: reserved

Bit 31 is msb; 0 is lsb.

**Comments:** To optimize performance of the Profile real-time software, a timecode input is not “read” unless it is connected to another resource. This way, cycles aren’t wasted for multiple timecode inputs that aren’t being used.

## VdrGetCurrentUserBits

**#include "tekvdr.h"**

**UINT VdrGetCurrentUserBits**(**ResourceHandle** *resourceHandle*)

This function returns the current user bits from the given timecode resource.

**Parameters:**

*resourceHandle:* The handle to one of the following:

- ResourceLtcInput
- ResourceVitcInput
- ResourceGenTcInput
- ResourceTimecodeRecorder

**Return Value:** This function returns a 32-bit unsigned integer, user bits.

## VdrGetEncodingRange

**#include <tekvdr.h>**

**BOOL VdrGetEncodingRange(ResourceHandle** *resourceHandle*,
**UINT*** *firstEncodedLine*,
**UINT*** *lastEncodedLine*,
**BOOL*** *field1OnTop*)

Gets the range of video lines to be encoded.

**Parameters:**

*resourceHandle:* Resource handle returned by *VdrAllocateResource* for the resource type ResourceMpegEncoder.

*firstEncodedLine:* Pointer to variable to receive the first video line to be encoded (inclusive).

*lastEncodedLine:* Pointer to variable to receive the line after the last video line to be encoded.

*field1OnTop:* Pointer to variable to receive TRUE if field 1 is interlaced on top; FALSE if field 2 is interlaced on top.

**Return Value:** TRUE if successful; FALSE otherwise.

**Comments:** Any pointer value of *firstEncodedLine*, *lastEncodedLine*, or *field1OnTop* which is NULL will have no value returned.

**See Also:** *VdrSetEncodingRange* on page 422.

## VdrGetFieldDominance

**#include "tekvdr.h"**

**FldDominanceMode VdrGetFieldDominance(VdrHandle** *vdrHandle*)

This function returns one of the following field dominance modes:

| | |
|---|---|
| FldDominanceNone: | No field dominance. |
| Fld1Dominant: | Field-1 dominance. |
| Fld2Dominant: | Field-2 dominance. |

**Parameters:**

*vdrHandle*: The handle returned by *VdrSetFieldDominance*.

**Return Value:** This function returns the current field dominance mode.

## VdrGetGenTcFormat

**#include "tekvdr.h"**

**TcFormat VdrGetGenTcFormat(ResourceHandle** *resourceHandle*)

Returns the format used for generating timecode by the timecode generator represented by resourceHandle.

**Parameters:**

*resourceHandle:* The value returned by *VdrAllocateResource*.

**Return Value:** Format used for generating timecode by the timecode generator represented by *resourceHandle*.

## VdrGetGenTcMode

**#include "tekvdr.h"**

**TcMode VdrGetGenTcMode(ResourceHandle** *resourceHandle*)

This function returns the timecode generator mode for the given generator resource.

**Parameters:**

*resourceHandle:* The handle to **ResourceGenTcInput**.

**Return Value:** This function returns the timecode generator mode for the given generator resource.

## VdrGetMajorVersion

**#include "tekvdr.h"**

**UINT VdrGetMajorVersion**(**ConnectHandle** *connHandle*)

This function returns the major number within the DLL version. For example, the major number in version 1.23 is **1**.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the major version number.

## VdrGetMaxChrQ

**#include "tekvdr.h"**

**double VdrGetMaxChrQ(ResourceHandle** *resourceHandle*)

This function returns the maximum chrominance quantization level being used for the given codec resource.

**Parameters:**

*resourceHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the maximum chrominance quantization default level or the level set by *VdrSetMaxChrQ* or *VdrSetChrQLimits*.

## VdrGetMaxLumQ

**#include "tekvdr.h"**

**double VdrGetMaxLumQ(ResourceHandle** *resourceHandle*)

This function returns the maximum luminance quantization level being used for the given codec resource.

**Parameters:**

*resourceHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the maximum luminance quantization level set set by *VdrSetMaxLumQ* or *VdrSetLumQLimits*.

## VdrGetMaxPosition

**#include "tekvdr.h"**

**INT VdrGetMaxPosition(VdrHandle** *vdrHandle*)

This function returns the maximum field number which can be reached during playback or record when **MediaPlayMode** is not **PlayNormal**.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the maximum reference field number (for example, the maximum clock limit).

## VdrGetMediaAtPosition

**#include "tekvdr.h"**

**MediaHandle VdrGetMediaAtPosition(ResourceHandle** *resourceHandle*,
**INT** *fieldNumber*)

For the given resource handle, this function returns the media handle which is playing or recording at the given reference field position. For the returned media handle, the value returned by a call to *VdrGetMediaStartPosition* will be less than or equal to the given position, and *VdrGetMediaEndPosition* will be greater than the given position.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

*fieldNumber:* The reference field position.

**Return Value:** This function returns the media handle at the reference field position, or NULL if there are no media handles at that position.

## VdrGetMediaAudioFades

**#include "tekvdr.h"**

**BOOL VdrGetMediaAudioFades**(**MediaHandle** *mediaHandle*,
**double*** *upLevel*,
**UINT*** *upFields*,
**double*** *downLevel*,
**UINT*** *downFields*)

This function returns the audio fade information associated with the specified media handle.

**Parameters:**

| | |
|---|---|
| *mediaHandle:* | The handle returned by *VdrAttachMedia*. |
| *upLevel:* | The level--from `0.0` (silence) to `8.0` (maximum gain)--at which the audio segment is to be played. |
| *upFields:* | The number of video fields over which to ramp the audio level to the final playback level. |
| *downLevel:* | The level--from `0.0` (silence) to `8.0` (maximum gain)--at which the audio segment will be terminated during playback. |
| *downFields:* | The number of fields over which to ramp the end of the audio to its termination. The ramp will start this many video fields from the end of the audio segment on the timeline. |

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** Each piece of audio media on the timeline may be associated with fade operations that establish ramps:

- between the level of the audio prior to the start of the segment playback and the playback level; and
- between the playback level and a level at the end of the segment.

The change at the front end of the segment is the *upramp*, described by the up parameters as a target level for playback and the number of fields over which the audio should be ramped to the target level. The change at the end of the segment is the *downramp*, described by the down parameters as the cut-off level of the audio at the end of the media segment playback and the number of fields over which to ramp down to the cut-off level. The downramp will start downFields before the end of the audio segment. The upramp starts with the first field of the audio segment. The level is specified as a multiplier of the audio sample's numeric value , from `0.0` to `8.0`, with `0` producing silence, `1` producing unity gain, and `8` producing the maximum audio gain of the audio board.

**See Also:** *VdrSetMediaAudioFades* on page 435.

## VdrGetMediaDuration

**#include "tekvdr.h"**

**UINT VdrGetMediaDuration**(**MediaHandle** *mediaHandle*)

This function allows the caller to query for the current duration of a piece of media. The existing call, *VdrGetMediaMarkDuration*, remains unchanged and returns the out-point/in-point.

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

**Return Value:** This function returns the current duration of a piece of media, otherwise, the function returns NULL.

**Comments:** The timeline duration fitting APIs allow automated control of timeline playback by “fitting” pieces of media onto the timeline for a given duration. The video subsystem will playback the timeline segments based on how many actual fields of video spanned the given duration. The formula is “Mark-out (or Mark-in) / duration”. This will occur automatically while each individual piece of media is played.

This function also allows scheduling (with new audio) of spots, in a commercial spot insertion environment, to “fit" an actual scheduled duration without having to constantly monitor and affect rate changes.

## VdrGetMediaEndPosition

**#include "tekvdr.h"**

**INT VdrGetMediaEndPosition**(**MediaHandle** *mediaHandle*)

This function returns the reference clock position corresponding to the field after the last field to be played or recorded in the given media handle.

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

**Return Value:** This function returns the reference field position at which media handle ends.

## VdrGetMediaMarkDuration

**#include "tekvdr.h"**

**UINT VdrGetMediaMarkDuration**(**MediaHandle** *mediaHandle*)

This function returns the duration, in fields, between the in-point and out-point for the given attached media file.

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

**Return Value:** This function returns the duration from the in-point and out-point for the media handle.

## VdrGetMediaMarkIn

**#include "tekvdr.h"**

**INT VdrGetMediaMarkIn**(**MediaHandle** *mediaHandle*)

This function returns the in-point for the given media handle.

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

**Return Value:** This function returns the in-point for the media handle.

## VdrGetMediaMarkOut

**#include "tekvdr.h"**

**INT VdrGetMediaMarkOut**(**MediaHandle** *mediaHandle*)

This function returns the out-point for the given media handle.

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

**Return Value:** This function returns the out-point for the media handle.

## VdrGetMediaName

**#include "tekvdr.h"**

**BOOL VdrGetMediaName**(**MediaHandle** *mediaHandle*,
**UINT** *maxNameLength*,
**CHAR*** *name*)

This function copies the name of the media file represented by the given media handle to the given buffer. This will return an empty name ("") for media handles that are attached with a NULL or empty media file path.

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

*maxNameLength:* The maximum length of a name which can be returned, including the NULL termination character.

*name:* The pointer to the buffer in which the name is to be returned.

**Return Value:** TRUE if the media name was found; otherwise, FALSE.

## VdrGetMediaStartPosition

**#include "tekvdr.h"**

**INT VdrGetMediaStartPosition**(**MediaHandle** *mediaHandle*)

This function returns the reference clock position at which the given media handle will begin to play or record.

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

**Return Value:** This function returns the reference field position at which the media handle starts.

## VdrGetMediaUserData

**#include "tekvdr.h"**

**void* VdrGetMediaUserData**(**MediaHandle** *mediaHandle*)

For the given media handle, this function returns a pointer to a block of user data. This pointer is set by a call to *MediaUserData*.

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

**Return Value:** This function returns the pointer to user data memory or NULL if *VdrSetMediaUserData* was never called for the given media handle.

## VdrGetMinChrQ

**#include "tekvdr.h"**

**double VdrGetMinChrQ(ResourceHandle** *resourceHandle*)

This function returns the minimum chrominance quantization level for the given codec resource.

**Parameters:**

*resourceHandle:* The handle to a resource of type ResourceJpegCodec returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the minimum chrominance quantization default level or the level set by *VdrSetMinChrQ* or *VdrSetChrQLimits*.

## VdrGetMinLumQ

**#include "tekvdr.h"**

**double VdrGetMinLumQ(ResourceHandle** *resourceHandle*)

This function returns the minimum luminance quantization level for the given codec resource.

**Parameters:**

*resourceHandle:* The handle to a resource of type ResourceJpegCodec returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the minimum luminance quantization default level or the level set by *VdrSetMinLumQ* or *VdrSetLumQLimits*.

## VdrGetMinorVersion

**#include "tekvdr.h"**

**UINT VdrGetMinorVersion**(**ConnectHandle** *connHandle*)

This function returns the minor number within the DLL version. For example, the minor number in version 1.23 is **23**.

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the minor version number.

## VdrGetMinPosition

**#include "tekvdr.h"**

**INT VdrGetMinPosition(VdrHandle** *vdrHandle*)

This function returns the minimum field number which can be reached during playback or record when **MediaPlayMode** is not **PlayNormal**.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the minimum reference field number (for example, the minimum clock limit).

## VdrGetMovieAccessMode

**#include "tekvdr.h"**

**MediaAccessMode VdrGetMovieAccessMode(MovieHandle** *movieHandle*)

This function returns the access mode for the given attached movie.

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

**Return Value:** This function returns one of the following access modes:

- AccessReadOnly
- AccessReadWrite

## VdrGetMovieAtPosition

**#include "tekvdr.h"**

**MovieHandle VdrGetMovieAtPosition**(**ResourceHandle** *resourceHandle*,
**INT** *fieldNumber*)

For the given resource handle, this function returns the movie handle which is playing or recording at the given reference field position. For the returned movie handle, the value returned by a call to *VdrGetMovieStartPosition* will be less than or equal to the given position and *VdrGetMovieEndPosition* will be greater than the given position.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.
*fieldNumber:* The reference field position.

**Return Value:** This function returns the movie at the selected field position.

## VdrGetMovieDefaultMarkIn

**#include "tekvdr.h"**

**UINT VdrGetMovieDefaultMarkIn**(**MovieHandle** *movieHandle,*
**ResourceHandle** *resourceHandle*)

For the given movie handle attached to the given resource handle, this function returns the default in-point; this represents the first recorded field.

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

*resourceHandle:* The handle returned by *VdrAllocateResource*.

**Return Value:** This function returns the default in-point.

## VdrGetMovieDefaultMarkOut

**#include "tekvdr.h"**

**UINT VdrGetMovieDefaultMarkOut**(**MovieHandle***movieHandle*,
**ResourceHandle** *resourceHandle*)

For the given movie handle attached to the given resource handle, this function returns the default out-point. This represents the field after the last recorded field.

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

*resourceHandle:* The handle returned by *VdrAllocateResource*.

**Return Value:** This function returns the movie handle returned by *VdrAttachMovie*.

## VdrGetMovieEndPosition

**#include "tekvdr.h"**

**INT VdrGetMovieEndPosition(MovieHandle** *movieHandle*,
**ResourceHandle** *resourceHandle*)

This function returns the reference clock position at the field following the last field played or recorded in the given movie handle.

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

*resourceHandle:* The handle returned by *VdrAllocateResource*.

**Return Value:** This function returns the end reference field.

## VdrGetMovieMarkDuration

**#include "tekvdr.h"**

**UINT VdrGetMovieMarkDuration**(**MovieHandle** *movieHandle*)

This function returns the duration (in fields) between the in- and out-points for the given movie.

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

**Return Value:** This function returns the duration from the in-point to the out-point.

## VdrGetMovieMarkIn

**#include "tekvdr.h"**

**INT VdrGetMovieMarkIn(MovieHandle** *movieHandle*)

This function returns the in-point for the given movie handle. This is the first field to be played or recorded for the given attached movie.

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

**Return Value:** This function returns the in-point for movie handle.

## VdrGetMovieMarkOut

**#include "tekvdr.h"**

**INT VdrGetMovieMarkOut(MovieHandle** *movieHandle*)

This function returns the out-point for the given movie handle. This is the field after the last field to be played or recorded for the given attached movie.

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

**Return Value:** This function returns the out-point for the movie handle.

## VdrGetMovieName

**#include "tekvdr.h"**

**BOOL VdrGetMovieName**(**MovieHandle** *movieHandle*,
**UINT** *maxNameLength*,
**CHAR*** *name*)

This function returns the name of the given attached movie.

**Parameters:**

| | |
|---|---|
| *movieHandle:* | The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*. |
| *maxNameLength:* | Maximum number of characters in the movie name. |
| *name:* | The name of the movie found. This must be a buffer of at least *maxNameLength*+1 characters. |

**Return Value:** This function returns the name of the movie.

## VdrGetMovieStartPosition

**#include "tekvdr.h"**

**INT VdrGetMovieStartPosition**(**MovieHandle** *movieHandle*,
**ResourceHandle** *resourceHandle*)

This function returns the reference clock position at which the given movie handle will begin to play or record.

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

*resourceHandle:* The handle returned by *VdrAllocateResource*.

**Return Value:** This function returns the start reference field.

## VdrGetMovieUserData

**#include "tekvdr.h"**

**void* VdrGetMovieUserData**(**MovieHandle** *movieHandle*)

For the given movie handle, this function returns a pointer to a block of user data. This pointer is set by a call to *VdrSetMovieUserData*.

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

**Return Value:** This function returns the pointer to a block of user data.

## VdrGetMpegChromaFormat

**#include "tekvdr.h"**

**MpegChromaFmt VdrGetMpegChromaFormat**(**ResourceHandle** *resourceHandle*)

Returns the chroma format currently used to encode video on the given resource handle.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

**Return Value:** The chroma format for the MPEG encoder resource, MpegChroma420 or MpegChroma422.

**See Also:** *VdrSetMpegChromaFormat* on page 446.

## VdrGetMpegGopLimits

**#include "tekvdr.h"**

**BOOL VdrGetMpegGopLimits**(**ConnectHandle** *connHandle,*
**UINT*** *iPixPerGop,*
**UINT*** *pPixPerIPix,*
**UINT*** *bPixPerIPPix*)

This call returns the maximum GopLimits for *IPixPerGop*, *pPixPerIPix*, and *bPixPerIPPix*.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *iPixPerGop:* | Pointer to a variable to receive the maximum allowable number of pictures per group of pictures. |
| *pPixPerIPix:* | Pointer to a variable to receive the maximum allowable number of P-pictures generated per I-picture. |
| *bPixPerIPPix:* | Pointer to a variable to receive the maximum allowable number of B-pictures generated per anchor picture. |

**Return Value:** TRUE if the call succeeds, FALSE otherwise.

## VdrGetMpegGopStructure

**#include "tekvdr.h"**

**BOOL VdrGetMpegGopStructure**(**ResourceHandle** *resourceHandle,*
**MpegGopEnd*** *mpegGopEnd,*
**UINT*** *reserved1,*
**UINT*** *pPixPerIPix,*
**UINT*** *bPixPerIPPix,*
**UINT*** *reserved2*)

Get the group of pictures (GOP) structure used to encode video on the given resource handle.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

*mpegGopEnd*: Pointer to a variable to receive type of GOP end, GopOpenEnd or GopClosedEnd.

*reserved1:* Set this value to NULL.

*pPixPerIPix:* Pointer to a variable to receive number of P-pictures generated per I-picture.

*bPixPerIPPix:* Pointer to a variable to receive number of B-pictures generated per anchor picture.

*reserved2:* Set this value to NULL.

**Return Value:** TRUE if the call succeeds, FALSE otherwise.

**Comments:** Passing NULL for any pointer value will skip any attempt to return the associated value; this does not affect the returned (TRUE/FALSE) value.

**See Also:** *VdrSetMpegGopStructure* on page 447.

## VdrGetNextEvent

**#include "tekvdr.h"**

**EventHandle VdrGetNextEvent(VdrHandle** *vdrHandle*,
**EventHandle** *fromEventHandle*)

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.
*fromEventHandle:* The pointer to the handle returned by *VdrScheduleEvent*.

**Return Value:** This function returns the next event on the timeline. If NULL, the function returns the first event.

**See Also:** *VdrGetPreviousEvent* on page 380.

## VdrGetNextMedia

**#include "tekvdr.h"**

**MediaHandle VdrGetNextMedia**(**ResourceHandle** *resourceHandle*,
**MediaHandle** *fromMediaHandle*)

This function finds the next media handle attached to the given resource handle after the *fromMediaHandle* or the first media handle if *fromMediaHandle* is NULL. Media handles are returned in the same order that would occur during forward motion play.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.
*fromMediaHandle:* The pointer to the handle returned by *VdrAttachMedia* or NULL.

**Return Value:** This function returns the next media handle or NULL if none is found.

## VdrGetNextMovie

**#include "tekvdr.h"**

**MovieHandle VdrGetNextMovie**(**ResourceHandle** *resourceHandle*,
**MovieHandle** *fromMovieHandle*)

This function finds the next movie handle attached to the given resource handle after the *fromMovieHandle* or the first media handle if *fromMovieHandle* is NULL. Movie handles are returned in the same order that would occur during forward motion play.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.
*fromMediaHandle:* The pointer to the handle returned by *VdrAttachMedia* or NULL.

**Return Value:** This function returns the next media handle or NULL if none is found.

## VdrGetNextPort

**#include "tekvdr.h"**

**VdrHandle VdrGetNextPort**(**ConnectHandle** *connHandle*,
**VdrHandle** *vdrHandle*)

This function returns the next port handle.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |

**Return Value:** This function returns the next port. If NULL, the function returns the first port.

**See Also:** *VdrGetPreviousPort* on page 383.

## VdrGetNextResource

**#include "tekvdr.h"**

**ResourceHandle VdrGetNextResource(VdrHandle** *vdrHandle*,
**ResourceHandle** *fromResourceHandle*)

This function finds the next resource handle attached to the given port handle after the *fromResourceHandle* or the first resource handle if *fromResourceHandle* is NULL. The returned handles have been previously allocated by calls to *VdrAllocateResource*.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*fromResourceHandle:* The pointer to the handle returned by *VdrAllocateResource* or NULL.

**Return Value:** This function returns the handle of the next resource or NULL if there is no next resource.

## VdrGetNextResourceOfType

**#include "tekvdr.h"**

**ResourceHandle VdrGetNextResourceOfType(VdrHandle** *vdrHandle*,
**ResourceHandle** *fromResourceHandle*,
**ResourceType** *resourceType*)

This function finds the next resource handle of the given resource type attached to the given port handle after the *fromResourceHandle* or the first resource handle if *fromResourceHandle* is NULL. The returned handles have been previously allocated by calls to *VdrAllocateResource*.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*fromResourceHandle:* The pointer to the handle returned by *VdrAllocateResource* or NULL.

*resourceType:* The type of resource to get. (See *Table 20, Vdr resource types* on page 283.)

**Return Value:** This function returns the handle of the next resource of the given type, or NULL if there is no next resource.

## VdrGetNumChannelDefs

**#include "tekvdr.h"**

**INT VdrGetNumChannelDefs(ConnectHandle** *connHandle* )

This function returns the number of channel definitions that exist on the Profile.

**Parameters:**

| | |
|---|---|
| *connHandle* | The value returned from RemOpenConnection or LOCAL_CONNECTION. |
| *Return value:* | The number of channel definitions. A negative value indicates that an error occurred while trying to access the channel definitions. |

## VdrGetPlayMode

**#include "tekvdr.h"**

**MediaPlayMode VdrGetPlayMode(VdrHandle** *vdrHandle*)

This function returns the play mode for the given port. See *VdrSetPlayMode* for details.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the play mode for the given port.

## VdrGetPortFromEvent

**#include "tekvdr.h"**

**VdrHandle VdrGetPortFromEvent(EventHandle** *eventHandle*)

This function returns the handle to the port on which the given event is scheduled.

**Parameters:**

*eventHandle:* The handle to the event returned by *VdrScheduleEvent*.

**Return Value:** This function returns the handle to the port on which the given event is scheduled.

## VdrGetPortFromMedia

**#include "tekvdr.h"**

**VdrHandle VdrGetPortFromMedia**(**MediaHandle** *mediaHandle*)

This function returns the handle to the port owning the resource on which the given media was attached.

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

**Return Value:** This function returns the handle to the port owning the resource on which the given media was attached.

## VdrGetPortFromMovie

**#include "tekvdr.h"**

**VdrHandle VdrGetPortFromMovie(MovieHandle** *movieHandle*)

This function finds the handle to the port owning the resources on which the given movie was attached.

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

**Return Value:** This function returns the handle to the port owning the resources on which the given movie was attached.

## VdrGetPortFromResource

**#include "tekvdr.h"**

**VdrHandle VdrGetPortFromResource**(**ResourceHandle** *resourceHandle*)

This function finds the handle to the port to which the given resource is allocated.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

**Return Value:** This function returns the handle to the port to which the given resource is allocated.

## VdrGetPosition

**#include "tekvdr.h"**

**INT VdrGetPosition**(**VdrHandle** *vdrHandle*)

This function finds the current position (reference field number).

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns the current position.

## VdrGetPreviousEvent

**#include "tekvdr.h"**

**EventHandle VdrGetPreviousEvent(VdrHandle** *vdrHandle*,
**EventHandle** *fromEventHandle*)

This function finds the previous event on the timeline, or the last event if *fromEventHandle* is NULL.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*fromEventHandle:* The pointer to the handle returned by *VdrScheduleEvent*.

**Return Value:** This function returns the previous event handle or NULL if none is found.

## VdrGetPreviousMedia

**#include "tekvdr.h"**

**MediaHandle VdrGetPreviousMedia(ResourceHandle** *resourceHandle*,
**MediaHandle** *fromMediaHandle*)

This function finds the previous media handle attached to the given resource handle before the *fromMediaHandle* or the last media handle if *fromMediaHandle* is NULL. Media handles are returned in the reverse order that would occur during forward motion play.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.
*fromMediaHandle:* The pointer to the handle returned by *VdrAttachMedia* or NULL.

**Return Value:** This function returns the previous media handle or NULL if none is found.

## VdrGetPreviousMovie

**#include "tekvdr.h"**

**MovieHandle VdrGetPreviousMovie**(**ResourceHandle** *resourceHandle*,
**MovieHandle** *fromMovieHandle*)

This function finds the previous movie handle attached to the given resource handle before the *fromMovieHandle*, or the last movie handle if *fromMovieHandle*is **NULL**. Movie handles are returned in the reverse order that would occur during forward motion play.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.
*Frommoviehandle:*The handle returned by *VdrAttachMovie*.

**Return Value:** This function returns the previous movie handle or NULL if none was found.

## VdrGetPreviousPort

**#include "tekvdr.h"**

**VdrHandle VdrGetPreviousPort(ConnectHandle** *connHandle*,
**VdrHandle** *vdrHandle*)

This function finds the previous port handle attached to the given *vdrHandle*.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |

**Return Value:** This function returns the previous port handle or Null if none is found.

## VdrGetPreviousResource

**#include "tekvdr.h"**

**ResourceHandle VdrGetPreviousResource(VdrHandle** *vdrHandle*,
**ResourceHandle** *fromResourceHandle*)

This function finds the previous resource handle attached to the given port handle before the *fromResourceHandle* or the last resource handle if *fromResourceHandle* is NULL. The returned handles have been previously allocated by calls to *VdrAllocateResource*.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*fromResourceHandle:* The pointer to the handle returned by *VdrAllocateResource*.

**Return Value:** This function returns the handle of the previous resource or NULL if there is no previous resource.

## VdrGetPreviousResourceOfType

**#include "tekvdr.h"**

**ResourceHandle VdrGetPreviousResourceOfType(VdrHandle** *vdrHandle*,
**ResourceHandle** *fromResourceHandle*,
**ResourceType** *resourceType*)

This function finds the previous resource handle attached to the given port handle before the *fromResourceHandle* or the last resource handle if *fromResourceHandle* is NULL. The returned handles have been previously allocated by calls to *VdrAllocateResource*.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*fromResourceHandle:* The pointer to the handle returned by *VdrAllocateResource* or NULL.

*resourceType:* The type of resource to get. (See *Table 20*, *Vdr resource types* on page 283.)

**Return Value:** This function returns the handle of the previous resource of the given type or NULL if there is no previous resource.

## VdrGetRecordCapacity

**#include "tekvdr.h"**

**UINT VdrGetRecordCapacity**(**VdrHandle** *vdrHandle*,
**UINT** *reserved*,
**UINT** *driveNumber*)

This function returns the amount of disk space (in fields) available for recording.The amount returned is the total amount available to all ports in the system. This value is only an approximation. The actual number of fields you can record may be greater or fewer than the number returned.

**Parameters:**

| | |
|---|---|
| *vdrHandle* | The handle returned by *VdrOpenPortConnection*. |
| *reserved:* | This value is ignored. |
| *driveNumber:* | The logical drive number. |

**Return Value:** This function returns the approximate number of fields remaining.

***NOTE: The GoalSize parameter is a legacy parameter that is still reserved but is currently not used.***

## VdrGetResourceConnectionHandle

**#include "tekvdr.h"**

**ResourceHandle VdrGetResourceConnectionHandle(VdrHandle** *vdrHandle,*
**ResourceType** *resourceType,*
**UINT** *resourceNumber*)

This function returns a handle which can be used for connection to a resource. The handle does not allow control or modification of the resource.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |
| *resourceType:* | The type of resource to be connected. See *VdrAllocateResource*. |
| *resourceNumber:* | Varies depending on type of resource and how the particular disk recorder is configured. |

**Return Value:** This function returns the handle used for connection to the resource.

**Comments:** When making a connection, the first handle (for the source) can be a value returned by *VdrAllocateResource* or by *VdrGetResourceConnectionHandle*. The second handle (for the destination) must be from *VdrAllocateResource*, otherwise one application could undo connections needed by another application.

## VdrGetResourceFromMedia

**#include "tekvdr.h"**

**ResourceHandle VdrGetResourceFromMedia**(**MediaHandle** *mediaHandle*)

This function returns the handle to the resource to which the given media handle is assigned.

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

**Return Value:** This function returns the handle for the resource to which media was attached.

## VdrGetResourceNumber

**#include "tekvdr.h"**

**UINT VdrGetResourceNumber**(**ResourceHandle** *resourceHandle*)

This function returns the number of the given resource. This number is the same number used in the call to *VdrAllocateResource*.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

**Return Value:** This function returns the number of the resource.

## VdrGetResourceType

**#include "tekvdr.h"**

**ResourceType VdrGetResourceType(ResourceHandle** *resourceHandle*)

This function returns the type of the given resource. This type is the same type used in the call to *VdrAllocateResource*.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

**Return Value:** This function returns the type of the resource.

## VdrGetResourceUserData

**#include "tekvdr.h"**

**void* VdrGetResourceUserData(ResourceHandle** *resourceHandle*)

For the given resource handle, this function returns a pointer to a block of user data. This pointer is set by a call to *VdrSetResourceUserData*.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

**Return Value:** This function returns the pointer to the block of user data or NULL.

## VdrGetSavedVbiLines

**#include "tekvdr.h"**

**BOOL VdrGetSavedVbiLines(ResourceHandle** *resourceHandle*,
**UINT** *maxNumLumaLines*,
**UINT*** *lumaLineNumbers*,
**UINT*** *numLumaLinesSaved*,
**UINT** *maxNumChromaLines*,
**UINT*** *chromaLineNumbers*,
**UINT*** *numChromaLinesSaved*)

This function retrieves the numbers of the VBI lines that are set to be losslessly recorded when recording MPEG video.

**Parameters:**

*resourceHandle*: The handle to an MPEG encoder, from *VdrAllocateResource*. For future compatibility, this function will accept handles to media encoders which will also provide this functionality.

*maxNumLumaLines:* The maximum number of entries that can be stored in the array pointed to by *lumaLineNumbers*.

*lumaLineNumbers*: A pointer to an array of UINTs into which will be stored the video line numbers for which luminance data will be saved (losslessly recorded). The numbering follows the conventions in the CCIR-601 specification.

*numLumaLinesSaved*: The number of lines of luminance data being saved. This could be greater than the size of the *lumaLineNumbers* array if the specified array is not large enough to hold all of the lines that are set.

*maxNumChromaLines*: The maximum number of entries that can be stored in the array pointed to by *chromaLineNumbers*.

*chromaLineNumbers*: A pointer to an array of UINTs into which will be stored the video line numbers for which chrominance data will be saved (losslessly recorded). The numbering follows the conventions in the CCIR-601 specification.

*numChromaLinesSaved*: The number of lines of chrominance data being saved. This could be greater than the size of the *chromaLineNumbers* array if the specified array is not large enough to hold all of the lines that are set.

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** The maximum number of lines that can be saved (numLumaLinesSaved + numChromaLinesSaved) is 3 on the PDR series and 6 on the XP series of Profiles.

**See Also:** *VdrSetSavedVbiLines* on page 454.

## VdrGetStateChangeLatency

**#include "tekvdr.h"**

**UINT VdrGetStateChangeLatency(VdrHandle** *hVdr*)

This function returns the number of fields in advance of a desired state change that a command must be issued before it takes effect at the output.

**Parameters:**

*hVdr:* The handle returned by *VdrAllocateChannel* or *VdrOpenPortConnection*.

**Return value:** The value returned is VDR_INVALID_LATENCY_VALUE if an error occurred; call GetLastError to get more information about the cause of the error. Otherwise, it is the amount of time, expressed in fields, required from calling an API function which changes the state of the VDR port to when the action will take place.

**Comments:** The system latency value applies to the following API commands: *VdrJog*, *VdrShuttle* and *VdrIdle*, *VdrCuePlay* and *VdrCueRecord*. Thus, the states affected are: Jog play, Jog record, Shuttle play, Shuttle record, Cue play, Cue record, and Idle.

The latest time that *VdrShuttleAtGenTc* and *VdrStateEventAtGenTc* can be issued must be the system latency plus one frame prior to the desired action time.

**See Also:** *VdrShuttleAtGenTc* on page 460,
*VdrStateEventAtGenTc* on page 463.

## VdrGetStillPlayMode

**#include "tekvdr.h"**

**StillMode VdrGetStillPlayMode**(**VdrHandle** *vdrHandle*)

This function returns the playback mode to be used during still play. See *VdrSetStillPlayMode* for details.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns one of the following:

- PlayByField
- PlayByFrame

## VdrGetTotalRecordCapacity

**#include "tekvdr.h"**

**UINT VdrGetTotalRecordCapacity(VdrHandle** *vdrHandle*,
**UINT** *reserved*,
**UINT** *driveNumber*)

Returns the total record capacity, in fields, of the given drive number, in the context of the resource configuration for the port represented by *vdrHandle*.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | Value returned by *VdrOpenPortConnection*. |
| *reserved:* | Reserved value, must be `0`. |
| *driveNumber:* | The ordinal number of the drive whose capacity is being tested. Must be less than the value returned by *CfgGetNumFileSystems*. |

**Return Value:** Total record capacity, in fields, of the given drive number, in the context of the resource configuration for the port represented by *vdrHandle*.

**Comments:** The total record capacity is the recording capacity of the disk when it is newly formatted (no media or movies recorded). The record capacity is calculated based on the resources allocated to *vdrHandle* and how they are configured. For example, a larger goal size for an allocated video codec implies a smaller total record capacity.

## VdrGetVideoCodecTiming

**#include "tekvdr.h"**

**Comments:** New command; engineering specification not yet available.

## VdrGetVideoCombinerPartHandle

**#include "tekvdr.h"**

**ResourceHandle VdrGetVideoCombinerPartHandle(ResourceHandle** *resourceHandle*, **VideoCombinerPartId** *partId*)

This function returns the main handle or a handle to a specified input or output of the given composite resource.

**Parameters:**

*resourceHandle:* A handle that refers to a video combiner resource, returned from *VdrAllocateResource* or from a previous call to *VdrGetVideoCombinerPartHandle*.

*partId:* One of the enumerated types that specify the desired input or output, or that specifies the main handle for the resource.

**Return Value:** Returns the desired handle, or NULL if an error occurred. (For extended error information, use **GetLastError**.)

## VdrGetVideoFieldCount

**#include "tekvdr.h"**

**UINT VdrGetVideoFieldCount**(**ConnectHandle** *connHandle*)

Returns the total number of video fields since the genlock driver was started.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** Total number of video fields since the genlock driver was started.

**Comments:** In general, the genlock driver restarts at power-up. Return value wraps in 828.5 days (NTSC system) or 994.2 days (PAL system).

## VdrGetVideoFormat

**#include "tekvdr.h"**

**VideoFormat VdrGetVideoFormat(VdrHandle** *vdrHandle*)

This function returns the video format of the given Vdr port.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns one of the following:

- Format525_60Hz_2To1
- Format625_50Hz_2To1
- Format720_59_94Hz_1To1
- Format720_60Hz_1To1
- Format1080_24Hz_1To1
- Format1080_25Hz_2To1
- Format1080_29_97Hz_2To1
- Format1080_30Hz_2To1

All 720 and 1080 formats are SMPTE 292 video formats.

## VdrGetVideoGoalSize

**#include "tekvdr.h"**

**UINT VdrGetVideoGoalSize**(**ResourceHandle** *resourceHandle*)

This function returns the goal size for video compression (applies to recording only) for the given video codec resource.

**Parameters:**

*resourceHandle:* The handle to a video codec resource returned by *VdrAllocateResource*.

**Return Value:** This function returns the video goal size for the given codec resource.

## VdrIdle

**#include "tekvdr.h"**

**BOOL VdrIdle(VdrHandle** *vdrHandle*)

This function ceases playback or record of audio and video.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns TRUE if the channels entered idle mode, FALSE if they did not.

**Table 22. Vdr idle states**

| Current state | Command | Next state |
|---|---|---|
| idle | VdrShuttle | motion play |
| | VdrJog | still play |
| | VdrSetPosition | idle (new value is taken as initial starting position for record or play) |
| | VdrCuePlay | cue play |
| | VdrCueRecord | cue record |
| | VdrIdle | idle |
| cue record | VdrShuttle | motion recording |
| | VdrJog | still recording |
| | VdrSetPosition | cue record |
| | VdrCueRecord | cue record |
| | VdrCuePlay | cue play (nothing recorded) |
| | VdrIdle | idle (nothing recorded) |
| recording | VdrShuttle | motion recording |
| | VdrJog | still recording |
| | VdrSetPosition | recording |
| | VdrCueRecord | cue record |
| | VdrCuePlay | cue play |
| | VdrIdle | idle |
| cue play | VdrShuttle | motion play |
| | VdrJog | still play |
| | VdrSetPosition | cue play |
| | VdrCueRecord | cue record |
| | VdrCuePlay | cue play |
| | VdrIdle | idle |
| motion play | VdrShuttle | motion play |
| | VdrJog | still play |
| | VdrSetPosition | motion play (latency depends on position) |
| | VdrCueRecord | cue record |

**Table 22. Vdr idle states (Continued)**

| Current state | Command | Next state |
|---|---|---|
| | VdrCuePlay | cue play |
| | VdrIdle | idle |
| still play | VdrShuttle | motion play |
| | VdrJog | still play |
| | VdrSetPosition | still play |
| | VdrCueRecord | cue record |
| | VdrCuePlay | cue play |
| | VdrIdle | idle |

## VdrIsMoviePositionLimitingEnabled

**#include "tekvdr.h"**

**BOOL VdrIsMoviePositionLimitingEnabled(VdrHandle** *vdrHandle*)

This function reports the status of position limiting for the given VdrHandle.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns TRUE if position limiting is enabled. (Default value is TRUE.)

**See Also:** *VdrDisableMoviePositionLimiting* on page 311, *VdrEnableMoviePositionLimiting* on page 312.

## VdrIsResourceActive

**#include "tekvdr.h"**

**BOOL VdrIsResourceActive**(**ResourceHandle** *resourceHandle*)

This function returns TRUE if the given resource is active. This is determined by previous calls to *VdrActivateResource* and *VdrDeactivateResource*.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

**Return Value:** This function returns TRUE if the resource is active, FALSE if inactive.

## VdrIsResourceAvailable

**#include "tekvdr.h"**

**BOOL VdrIsResourceAvailable(VdrHandle** *vdrHandle*,
**ResourceType** *resourceType*,
**UINT** *resourceNumber*)

This function returns TRUE if the given resource type and number is available to be allocated via *VdrAllocateResource*.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |
| *resourceType:* | The resource type. (See *Table 20, Vdr resource types* on page 283.) |
| *resourceNumber:* | The resource number. (See *VdrAllocateResource*.) |

**Return Value:** This function returns TRUE if the given resource is available, FALSE otherwise.

## VdrJog

**#include "tekvdr.h"**

**BOOL VdrJog(VdrHandle** *vdrHandle*,
**INT** *numberOfFields*)

This function causes playback or recording to occur in still mode. If *numberOfFields* is non-zero, steps the image by the given number of fields. (See *VdrShuttle* for remarks on latency.)

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*numberOfFields:* The integer number of fields which the displayed image should move.

**Return Value:** This function returns TRUE if the playback or recording was set to jog or the given number of fields were jogged, FALSE if not.

## VdrMergeResources

**#include "tekvdr.h"**

**Comments:** New command; engineering specification not yet available.

## VdrOpenPortConnection

**#include "tekvdr.h"**

**VdrHandle VdrOpenPortConnection**(**ConnectHandle** *connHandle*)

This function opens a port to control the embedded disk recorder video and audio systems. This command must be called in order to obtain a port handle used with other commands.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the ConnectHandle if successful, Null if no port is available.

## VdrReleaseResource

**#include "tekvdr.h"**

**BOOL VdrReleaseResource**(**ResourceHandle** *resourceHandle*)

This function releases control of the given resource. The given resource handle cannot be used again after this call.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

**Return Value:** This function returns TRUE if the resource was released, FALSE if not.

**Comments:** A call to **ClosePort** will release all resource handles which have been obtained using *VdrAllocateResource* for that port. A call to *VdrReleaseResource* will detach all media handles obtained using *VdrAttachMedia* for the resource. The resource and media handles cannot be used after these calls.

## VdrRescheduleEvent

**#include "tekvdr.h"**

**BOOL VdrRescheduleEvent(INT** *atFieldNumber*,
**EventHandle** *eventHandle*)

This function reschedules the given event to occur at the given field number.

**Parameters:**

*atFieldNumber:* The field at which the event should occur.
*eventHandle:* The handle to the event returned by *VdrScheduleEvent*.

**Return Value:** This function returns TRUE if the event was rescheduled, FALSE if it was not.

## VdrResetPort

**#include "tekvdr.h"**

**BOOL VdrResetPort(VdrHandle** *vdrHandle*)

This function causes the embedded disk recorder system to delete all volatile data structures created from this Vdr port since the port was opened. Releases all resource handles and detaches the media.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

**Return Value:** This function returns TRUE if the port was reset, FALSE if it was not.

## VdrResetResource

**#include "tekvdr.h"**

**BOOL VdrResetResource**(**ResourceHandle** *resourceHandle*)

This function resets the given resource. Media handles attached to the resource will be detached, and corresponding media files will be detached from the timeline.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** The specified resource is reset to the factory default state. The application must ensure that the resource is correctly configured prior to further operation.

*VdrResetPort* is the preferred command to reset a resource.

**See Also:** *VdrResetPort* on page 411.

## VdrScheduleEvent

**#include "tekvdr.h"**

**EventHandle VdrScheduleEvent(VdrHandle** *vdrHandle,*
**INT** *atFieldNumber,*
**EventType** *type*)

This function will only take place while the port is in PlayReady or PlayActive state. It will never be effective during idle and record states.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |
| *atFieldNumber:* | The port timeline field number at which the event is scheduled to take effect. |
| *type:* | Specific event type and supporting parameters. (See *Table 23* below.) |

**Return Value:** This function returns the handle to the scheduled event; returns NULL if the event could not be scheduled.

**Comments:** The types of events listed in *Table 23* below can be used.

**See Also:** *VdrDefaultEvent* on page 304,
*VdrStateEvent* on page 462.

**Table 23. Vdr scheduled events**

| Event type | Description |
|---|---|
| **EventConnectResources** | At the given field number in forward play, connects the source resource to the destination resource. |
| Usage | (... **EventConnectResources**, **ResourceHandle** *sourceHandle*, **ResourceHandle** *destinationHandle*) |
| Arguments | *sourceHandle*: handle to a video or timecode resource providing output, returned by **VdrAllocateResource** or **VdrGetResourceConnectionHandle**.<br>*destinationHandle*: handle to a video or timecode resource requiring input, returned by **VdrAllocateResource**. |
| **EventMixAudio** | Starting at the given field number in forward play, mixes the output from the given audio codec to the given level at the given audio output, changing the mix ***at the specified field***. This means that the numFields parameter does not participate in the function. (Compare with definition in *Table 21* on page 305.) |
| Usage | (... **EventMixAudio**, **ResourceHandle** *audioCodecHandle*, **ResourceHandle** *audioOutputHandle*, **double** *mixLevel*, **UINT** *numFields*) |
| Arguments | *audioCodecHandle*: handle to the audio codec, returned by **VdrAllocateResource** or **VdrGetResourceConnectionHandle**.<br>*audioOutputHandle*: handle to the audio output, returned by **VdrAllocateResource**.<br>*mixLevel*: [0 ... 1.0]; level at which the output of the audio codec should be mixed into the audio output. 1.0 represents an unattentuated mix.<br>*numFields*: duration (in video fields) of the change in mix from the current mix to the given mix. |
| **EventSetGtcTime** | At the given field number in forward play, jams the specified timecode pattern into the specified timecode generator. |
| Usage | (... **EventSetGtcTime**, **ResourceHandle** *timeCodeGeneratorHandle*, **UINT** *pattern*) |
| Arguments | *timeCodeGeneratorHandle*: handle to timecode generator, returned by **VdrAllocateResource**.<br>*pattern*: timecode value encoded as four binary numbers, each a single-byte wide. **UINT** pattern = (hours << 24) \| (minutes << 16) \| (seconds << 8) \| (fields) |
| **EventSetGtcBits** | At the given field number in forward play, jams the user bit pattern into the timecode generator. |
| Usage | (... **EventSetGtcBits**, **ResourceHandle** *timeCodeGeneratorHandle*, **UINT** *bitpattern*) |
| Arguments | *timeCodeGeneratorHandle*: handle to timecode generator, returned by **VdrAllocateResource**.<br>*bitpattern*: As defined in proposed revised standard SMPTE 12M, April 1993. |

**Table 23. Vdr scheduled events (Continued)**

| Event type | Description |
|---|---|
| **EventVoiceOver** | Mixes the audio input at the given level to the audio output, changing the mix ***at the specified field***. This means that the numFields parameter does not participate in the function. (Compare with definition in *Table 21* on page 305.) |
| Usage | (... **EventVoiceOver**, **ResourceHandle** *audioInputHandle*, **ResourceHandle** *audioOutputHandle*, **double** *mixLevel*, **UINT** *numFields*)<br>This event is active when the port is in CuePlay or Play state. This event is only available with the ASPB audio board. |
| Arguments | *audioInputHandle*: handle to the audio input, returned by **VdrAllocateResource** or **VdrGetResourceConnectionHandle**.<br>*audioOutputHandle*: handle to the audio output, returned by **VdrAllocateResource**.<br>*mixLevel:* [.0 .. 1.0]; level at which the audio input should be mixed into the audio output. 1.0 represents an unattentuated mix.<br>*numFields*: duration (in video fields) of the change in mix from the current mix to the given mix. |

## VdrSetAudioInputLevel

**#include "tekvdr.h"**

**BOOL VdrSetAudioInputLevel(ResourceHandle** *resourceHandle*,
**double** *level*)

This function sets the level between the audio input (rear panel connection) and the audio codec. This affects the loudness of recorded material.

**Parameters:**

*resourceHandle:* The handle to **ResourceAudioCodec** returned by *VdrAllocateResource*.

*level:* [0.0 ... 8.0] audio level. The level is specified as a multiplier of the audio sample's numeric value , from `0.0` to `8.0`, with `0` producing silence (full attenuation), `1` producing unity gain (no attenuation and no gain), and `8` producing the maximum audio gain of the audio board.

**Return Value:** TRUE if successful; otherwise, FALSE.

## VdrSetAudioOutputLevel

**#include "tekvdr.h"**

**BOOL VdrSetAudioOutputLevel(ResourceHandle** *resourceHandle*,
**double** *level*)

This function sets the level between the audio codec and the audio output (rear panel connection); this affects the loudness of playback of previously-recorded material.

**Parameters:**

| | |
|---|---|
| *resourceHandle:* | The handle to **ResourceAudioCodec** returned by *VdrAllocateResource*. |
| *level:* | [0.0 ... 1.0] audio level. A value of `0.0` indicates full attenuation; `1.0` indicates no attenuation. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## VdrSetAudioWindow

**#include "tekvdr.h"**

**BOOL VdrSetAudioWindow**(**ResourceHandle** *resourceHandle*,
**UINT** *numberOfFields*)

This function sets the repetition interval (in fields) for the given audio resource when audio playback occurs during still play mode. The leakage frequency that appears in audio during still play is inversely related to the window size. This affects all attached audio media files for this audio resource. Volatile; default is 1.

**Parameters:**

*resourceHandle:* The handle to the audio codec returned by **AllocateResource**.

*numberOfFields:* The number of fields included in the repeated playback of audio during jog mode.

**Return Value:** This function returns TRUE if the audio jog length was set, FALSE if not.

## VdrSetBitRate

**#include "tekvdr.h"**

**BOOL VdrSetBitRate(ResourceHandle** *resourceHandle*,
**double** *bitRate*)

Set the bit-rate to be used for encoding.

**Parameters:**

| | |
|---|---|
| *resourceHandle:* | The handle returned by *VdrAllocateResource*. |
| *bitRate:* | Desired bit-rate for encoding, expressed in bits/second. For example, `bitRate=36.0e6` causes encoding at 36Mbps. |

**Return Value:** TRUE if the call succeeds; FALSE otherwise.

**See Also:** *VdrGetBitRate* on page 322.

## VdrSetChrQLimits

**#include "tekvdr.h"**

**BOOL VdrSetChrQLimits**(**ResourceHandle** *resourceHandle*,
**double** *minChrQ*,
**double** *maxChrQ*)

This function sets the minimum and maximum chroma quantization limits for the given video codec.

**Parameters:**

| | |
|---|---|
| *resourceHandle:* | The handle to **ResourceJpegCodec** returned by *VdrAllocateResource*. |
| *minChrQ:* | Minimum chroma Q. |
| *maxChrQ:* | Maximum chroma Q. |

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** The limits must be within the range given by *VdrGetAbsMinChrQ* and *VdrGetAbsMaxChrQ*.

## VdrSetDefaultTcFormat

**#include "tekvdr.h"**

**BOOL VdrSetDefaultTcFormat**(**VdrHandle** *vdrHandle*,
**UINT** *startingTc*,
**UNIT** *userBits*,
**TcFormat** *tcFormat*)

This function sets the default format for striping new timecode media files attached to the port represented by vdrHandle.

**Parameters:**

*vdrHandle:* The value returned by *VdrOpenPortConnection*.

*startingTc:* The starting timecode, as constructed by SET_TIMECODE. (See the SET_TIMECODE macro in *vdrtypes.h*)

*userBits:* User bits to be striped throughout the file. (See **EventSetGtcBits** for format.)

*tcFormat:* Timecode format for the timecode file, one of the following:

- TcFormatDropFrame
- TcFormatNonDropFrame

(There is no effect if the video format of the given file is `Format625_50Hz_2To1`).

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** All newly attached timecode files will be striped in the given format starting at 00:00:00:00. The video format used will be the video format of the port at the time the file is created.

## VdrSetEncodingRange

**#include <tekvdr.h>**

**BOOL VdrSetEncodingRange**(**ResourceHandle** *resourceHandle*,
**UINT** *firstEncodedLine*,
**UINT** *lastEncodedLine*,
**BOOL** *field1OnTop*)

Sets the range of video lines to be encoded.

**Parameters:**

*resourceHandle:* The resource handle returned by *VdrAllocateResource* for the resource type ResourceMpegEncoder.

*firstEncodedLine:* The first video line to be encoded.

*lastEncodedLine:* The line after the last video line to be encoded.

*field1OnTop:* Set to TRUE to interlace with field 1 on top, set to FALSE to interlace with field 2 on top.

**Return Value:** TRUE if successful, FALSE otherwise.

**Comments:** The total number of video lines encoded will be (lastEncodedLine - firstEncodedLine). However the IBM ME-30 and ME-31 encoders have a granularity of 32 lines per frame in the encoding range, so that the total number of lines will be increased if necessary so that the value is a multiple of 32. The minimum number of encoded lines is 480 in 525-line mode, and 576 in 625-line mode.

**See Also:** *VdrGetEncodingRange* on page 331.

## VdrSetFieldDominance

**#include "tekvdr.h"**

**BOOL VdrSetFieldDominance(VdrHandle** *vdrHandle*,
**FldDominanceMode** *dominance*)

This function sets the dominance mode for the given port to the given value. During recording, *VdrActivateResource* and *VdrDeactivateResource* also respect field dominance.

**Parameters:**

*vdrHandle:* The handle from *VdrOpenPortConnection*.

*dominance:* One of the following:

| | |
|---|---|
| FldDominanceNone: | No field dominance.<br>Recording begins or ends on either field 1 or field 2 (default). |
| Fld1Dominant: | Field-1 dominance.<br>Recording begins on field 1 and ends on field 2 (inclusive). |
| Fld2Dominant: | Field-2 dominance.<br>Recording begins on field 2 and ends on field 1 (inclusive). |

**Return Value:** The function returns TRUE if successful; FALSE otherwise.

## VdrSetFittedMediaMarkIn

**#include "tekvdr.h"**

**BOOL VdrSetFittedMediaMarkIn(MediaHandle** *mediaHandle*,
**INT** *inField*)

This function allows an in-point, on a piece of already attached media, to be modified. This function modifies the in-point while maintaining the existing duration of that particular piece. If the call *VdrSetMediaMarkIn* is used, the duration would be "scaled" to the appropriate amount to maintain the existing play rate (duration/Out, or duration/In).

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

*inField:* The number of the first field to be played or recorded.

**Return Value:** This function returns the modified in-point in a piece of attached media. Otherwise, the function returns NULL.

**Comments:** The timeline duration fitting APIs allow automated control of timeline playback by "fitting" pieces of media onto the timeline for a given duration. The video subsystem will playback the timeline segments based on how many actual fields of video spanned the given duration. The formula would be "mark-out (or mark-in) / duration". This will occur automatically while each individual piece of media is played.

This function also allows scheduling (with new audio) of spots, in a commercial spot insertion environment, to "fit" an actual scheduled duration without having to constantly monitor and affect rate changes.

## VdrSetFittedMediaMarkOut

**#include "tekvdr.h"**

**BOOL VdrSetFittedMediaMarkOut**(**MediaHandle** *mediaHandle,*
**INT** *outField*)

This function allows an out-point, on a piece of already attached media, to be modified. This function modifies the out-point while maintaining the existing duration of that particular piece. If the call *VdrSetMediaMarkOut* is used, the duration would be "scaled" to the appropriate amount to maintain the existing play rate (duration/Out, or duration/In).

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

*outField:* The number of the last field to be played or recorded.

**Return Value:** This function returns the modified out-point in a piece of attached media. Otherwise, the function returns NULL.

**Comments:** The timeline duration fitting APIs allow automated control of timeline playback by "fitting" pieces of media onto the timeline for a given duration. The video subsystem will playback the timeline segments based on how many actual fields of video spanned the given duration. The formula is "Mark-out (or Mark-in) / duration". This will occur automatically while each individual piece of media is played.

This function also allows scheduling (with new audio) of spots, in a commercial spot insertion environment, to "fit" an actual scheduled duration without having to constantly monitor and affect rate changes.

## VdrSetGenTcFormat

**#include "tekvdr.h"**

**BOOL VdrSetGenTcFormat**(**ResourceHandle** *resourceHandle*,
**TcFormat** *tcFormat*)

This function sets the timecode format for the given timecode generator.

**Parameters:**

*resourceHandle:* The handle to the resource of type **ResourceGenTcInput** returned by *VdrAllocateResource*.

*tcFormat:* One of the following:

- TcFormatDropFrame
- TcFormatNonDropFrame

**Return Value:** TRUE if successful; otherwise, FALSE.

## VdrSetGenTcMode

**#include "tekvdr.h"**

**BOOL VdrSetGenTcMode**(**ResourceHandle** *resourceHandle*,
**TcMode** *tcMode*)

This function sets the operation mode for the given timecode generator.

**Parameters:**

*resourceHandle:* The handle to the resource of type **ResourceGenTcInput** returned by *VdrAllocateResource*.

*tcMode:* One of the following:

- TcModeFreeze
- TcModeFreeRun
- TcModeFieldLocked

**Return Value:** TRUE if successful; otherwise, FALSE.

## VdrSetGenTcRegenSource

**#include "tekvdr.h"**

**BOOL VdrSetGenTcRegenSource**(**ResourceHandle** *genTcHandle*,
**ResourceHandle** *regenTcHandle*)

This function identifies the time source that should be used to synchronize the value in the timecode generator. The time source will normally be the VITC value on the HouseBurst black signal or the LTC value which are connected to the Reference GenLock connections of the Profile. The timecode generator should already be conditioned by the setting of its format and mode.

**Parameters:**

*genTcHandle:* The value returned by *VdrAllocateResource* for a timecode generator.
*regenTcHandle:* The value returned by *VdrAllocateResource* for the time source input.

**Return Value:** TRUE if the connection is successful; otherwise, FALSE.

## VdrSetGpiOutput

**#include "tekvdr.h"**

**BOOL VdrSetGpiOutput(ResourceHandle** *gpiOutputResource*,
**BOOL** *active*);

This function sets the specified GPI output to the active state.

**Parameters:**

*gpiOutputResource:* The handle to the resource of type **ResourceGpiOutput** returned by *VdrAllocateResource*.

*active:* TRUE for active, otherwise FALSE.

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** The output remains in the specified set state until explicitly changed by another command.

## VdrSetGpiOutputOnGpiInputTrigger

**#include "tekvdr.h"**

**BOOL VdrSetGpiOutputOnGpiInputTrigger(ResourceHandle** *gpiOutputResource*,
**BOOL** *active*,
**ResourceHandle** *gpiInputResource*);

This function sets the specified GPI output to the active state if a GPI trigger is detected on a specified input.

**Parameters:**

*gpiOutputResource:* The handle to the resource of type **ResourceGpiOutput** returned by *VdrAllocateResource*.

*active:* TRUE if active, otherwise FALSE.

*gpiInputResource:* The handle to the resource of type **ResourceGpiInput** returned by *VdrAllocateResource*.

**Return Value:** TRUE if successful; otherwise, FALSE.

## VdrSetLumQLimits

**#include "tekvdr.h"**

**BOOL VdrSetLumQLimits**(**ResourceHandle** *resourceHandle*,
**double** *minLumQ*,
**double** *maxLumQ*)

This function sets the minimum and maximum luminance quantization limits for the given video codec.

**Parameters:**

*resourceHandle:* The handle to the resource of type **ResourceJpegCodec** returned by *VdrAllocateResource*.

*minLumQ:* Minimum luminance Q.

*maxLumQ:* Maximum luminance Q.

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** The limits must be within the range given by *VdrGetAbsMinLumQ* and *VdrGetAbsMaxLumQ*.

## VdrSetMaxChrQ

**#include "tekvdr.h"**

**BOOL VdrSetMaxChrQ(ResourceHandle** *resourceHandle*,
**double** *chromaQ*)

This function sets the maximum chrominance quantization level for video compression with the given codec resource (applies to recording only).

**Parameters:**

*resourceHandle:* The handle returned by *VdrOpenPortConnection*.

*chromaQ:* Maximum chrominance Q.

**Return Value:** This function returns TRUE if the maximum chrominance quantization level was set, FALSE if not.

**Comments:** The maximum chrominance Q must be greater than the minimum chrominance Q and must be in the range given by *VdrGetAbsMinChrQ* and *VdrGetAbsMaxChrQ*.

## VdrSetMaxLumQ

**#include "tekvdr.h"**

**BOOL VdrSetMaxLumQ(ResourceHandle** *resourceHandle*,
**double** *lumQ*)

This function sets the maximum luminance quantization level for video compression with the given codec resource (applies to recording only).

**Parameters:**

*resourceHandle:* The handle returned by *VdrOpenPortConnection*.

*lumQ:* Maximum luminance Q.

**Return Value:** This function returns TRUE if the maximum luminance quantization level was set, FALSE if not.

**Comments:** The maximum luminance Q must be greater than the minimum luminance Q and must be in the range given by *VdrGetAbsMinLumQ* and *VdrGetAbsMaxLumQ*.

## VdrSetMaxPosition

**#include "tekvdr.h"**

**BOOL VdrSetMaxPosition(VdrHandle** *vdrHandle*,
**INT** *fieldNumber*)

This function sets the field after the last field number which can be reached during playback or record.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*maxFieldNumber***:** (Last field which can be played) + 1; the maximum length of the record or playback interval will be (*maxFieldNumber* - *minFieldNumber*).

**Return Value:** This function returns TRUE if the maximum field was set, FALSE if it was not.

**Comments:** The maximum position applies when **MediaPlayMode** is not **PlayNormal**. During **PlayNormal**, the maximum position is defined by the attached media out-points and the largest referenced field position of all out-points.

## VdrSetMediaAudioFades

**#include "tekvdr.h"**

**BOOL VdrSetMediaAudioFades**(**MediaHandle** *mediaHandle*,
**double** *upLevel*,
**UINT** *upFields*,
**double** *downLevel*,
**UINT** *downFields*)

This function sets the audio fade information on an audio media segment with the specified media handle.

**Parameters:**

| | |
|---|---|
| *mediaHandle:* | The handle returned by *VdrAttachMedia*. |
| *upLevel:* | The level--from `0.0` (silence) to `8.0` (maximum gain)--at which the audio segment is to be played. |
| *upFields:* | The number of video fields over which to ramp the audio level to the final playback level. |
| *downLevel:* | The level--from `0.0` (silence) to `8.0` (maximum gain)--at which the audio segment will be terminated during playback. |
| *downFields:* | The number of fields over which to ramp the end of the audio to its termination. The ramp will start this many video fields from the end of the audio segment on the timeline. |

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** Each piece of audio media on the timeline may be associated with fade operations that establish ramps:

- between the level of the audio prior to the start of the segment playback and the playback level; and
- between the playback level and a level at the end of the segment.

The change at the front end of the segment is the *upramp*, described by the up parameters as a target level for playback and the number of fields over which the audio should be ramped to the target level. The change at the end of the segment is the *downramp*, described by the down parameters as the cut-off level of the audio at the end of the media segment playback and the number of fields over which to ramp down to the cut-off level. The downramp will start downFields before the end of the audio segment. The upramp starts with the first field of the audio segment. The level is specified as a multiplier of the audio sample's numeric value , from `0.0` to `8.0`, with `0` producing silence, `1` producing unity gain, and `8` producing the maximum audio gain of the audio board.

**See Also:** *VdrGetMediaAudioFades* on page 340.

## VdrSetMediaDuration

**#include "tekvdr.h"**

**UINT VdrSetMediaDuration**(**MediaHandle** *mediaHandle*,
**UINT** *duration*,
**ShiftMode** *shiftMode*)

This function allows the caller to set the current duration of a piece of media. This is a new call to set the actual duration on the timeline.

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

*duration:* The length of the attached media in fields.

*shiftMode:* One of the following:

- ShiftBefore
- ShiftAfter

If *shiftMode* is **ShiftBefore**, the reference field numbers for fields in media files attached earlier in the list than *mediaHandle* will have a number of fields added equal to the length (in fields) of the detached media file. If *shiftMode* is **ShiftAfter**, the reference field numbers for fields in all media files attached later in the list than *mediaHandle* will have this number of fields subtracted.

**Return Value:** The function sets the current duration of a piece of media. Otherwise, the function returns NULL.

**Comments:** The timeline duration fitting APIs allow automated control of timeline playback by “fitting” pieces of media onto the timeline for a given duration. The video subsystem will playback the timeline segments based on how many actual fields of video span the given duration. The formula is “Mark-out (or Mark-in) / duration”. This will occur automatically while each individual piece of media is played.

This function also allows scheduling (with new audio) of spots, in a commercial spot insertion environment, to “fit” an actual scheduled duration without having to constantly monitor and affect rate changes.

## VdrSetMediaMarkIn

**#include "tekvdr.h"**

**BOOL VdrSetMediaMarkIn(MediaHandle** *mediaHandle,*
**INT** *inField,*
**ShiftMode** *shiftMode*)

This function sets the in-point for the given media handle for playback or recording. Volatile. The in value can be greater than the current out value. (The out value will slide to the same point as the new in value if it is greater.)

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

*inField:* The number of the first field to be played or recorded.

*shiftMode:* *One of the following:*

- ShiftBefore
- ShiftAfter

If *shiftMode* is **ShiftBefore**, the reference field numbers for fields in media files attached before *mediaHandle* will have (*inField* - *oldInField*) fields added, where *oldInField* was the value of the in-point before this command. If *shiftMode* is **ShiftAfter**, the reference field numbers for fields in media files attached after *mediaHandle* and the reference field numbers for fields in *mediaHandle* itself will have (*inField* - *oldInField*) fields subtracted.

**Return Value:** This function returns TRUE if the in-point was set, FALSE if not.

## VdrSetMediaMarkOut

**#include "tekvdr.h"**

**BOOL VdrSetMediaMarkOut**(**MediaHandle** *mediaHandle*,
**INT** *outField*,
**ShiftMode** *shiftMode*)

This function sets the out-point for the given media handle for playback, recording. Playback or recording will stop at the field prior to the out-point. Volatile. Default is (end of material + 1). (If the out value is set to less than the current in value, then the in value will slide to the new out value.)

**Parameters:**

*mediaHandle:* The handle returned by *VdrAttachMedia*.

*outField:* The number of the field following the last field to be played or recorded (unsigned integer).

*shiftMode:* *One of the following:*

- ShiftBefore
- ShiftAfter

If *shiftMode* is **ShiftBefore**, the reference field numbers for fields in media files attached before *mediaHandle* will have (*inField* - *oldInField*) fields added, where *oldInField* was the value of the in-point before this command. If *shiftMode* is **ShiftAfter**, the reference field numbers for fields in media files attached after *mediaHandle*, and the reference field numbers for fields in *mediaHandle* itself, will have (*inField* - *oldInField*) fields subtracted.

**Return Value:** This function returns TRUE if in-point was set, FALSE if not.

## VdrSetMediaUserData

**#include "tekvdr.h"**

**BOOL VdrSetMediaUserData(MediaHandle** *mediaHandle*,
**void*** *userData*)

For the given media handle, this function sets a pointer to a block of user data. This pointer can be recovered by a call to *VdrGetMediaUserData*.

**Parameters:**

| | |
|---|---|
| *mediaHandle:* | The handle returned by *VdrAttachMedia*. |
| *userData:* | The pointer to a block of user data. |

**Return Value:** This function returns TRUE if the pointer value attached to the media handle, FALSE otherwise.

## VdrSetMinChrQ

**#include "tekvdr.h"**

**BOOL VdrSetMinChrQ(ResourceHandle** *resourceHandle*,
**double** *chromaQ*)

This function sets the minimum chrominance quantization level for video compression with the given codec resource (applies to recording only).

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

*chromaQ:* Minimum chrominance Q.

**Return Value:** This function returns TRUE if pointer value attached to media handle, FALSE otherwise.

**Comments:** The minimum chrominance Q must be less than the maximum chrominance Q and must be in the range given by *VdrGetAbsMinChrQ* and *VdrGetAbsMaxChrQ*.

## VdrSetMinLumQ

**#include "tekvdr.h"**

**BOOL VdrSetMinLumQ**(**ResourceHandle** *resourceHandle*,
**double** *lumQ*)

This function sets the minimum luminance quantization level for video compression with the given codec resource (applies to recording only).

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

*lumQ:* Minimum luminance Q.

**Return Value:** This function returns TRUE if the minimum luminance quantization level was set, FALSE if not.

**Comments:** The minimum luminance Q must be less than the maximum luminance Q and must be in the range given by *VdrGetAbsMinLumQ* and *VdrGetAbsMaxLumQ*.

## VdrSetMinPosition

**#include "tekvdr.h"**

**BOOL VdrSetMinPosition(VdrHandle** *vdrHandle*,
**INT** *minFieldNumber*)

This function sets the minimum field number which can be reached during playback or record.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*minFieldNumber*: The minimum field which can be played. The maximum length of the record or playback interval will be (*maxFieldNumber* - *minFieldNumber*).

**Return Value:** This function returns TRUE if the minimum field was set, FALSE if it was not.

**Comments:** The minimum position applies when MediaPlayMode is not PlayNormal. During PlayNormal, the minimum position is defined by the attached media in-points and is the smallest referenced field position of all in-points.

## VdrSetMovieMarkIn

**#include "tekvdr.h"**

**BOOL VdrSetMovieMarkIn**(**MovieHandle** *movieHandle*,
**INT** *inField*,
**ShiftMode** *shiftMode*)

This function sets the in-point for the given movie handle for playback or recording.

**Parameters:**

| | |
|---|---|
| *movieHandle:* | The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*. |
| *inField:* | The first field to be played or recorded. |
| *shiftMode:* | See *VdrSetMediaMarkIn* for details. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## VdrSetMovieMarkOut

**#include "tekvdr.h"**

**BOOL VdrSetMovieMarkOut**(**MovieHandle** *movieHandle,*
**INT** *outField,*
**ShiftMode** *shiftMode*)

This function sets the out-point for the given movie handle for playback or recording.

**Parameters:**

| | |
|---|---|
| *movieHandle:* | The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*. |
| *outField:* | The field following the last field to be played or recorded. |
| *shiftMode:* | See *VdrSetMediaMarkIn* for details. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## VdrSetMovieUserData

**#include "tekvdr.h"**

**BOOL VdrSetMovieUserData(MovieHandle** *movieHandle*,
**void*** *userData*)

For the given movie handle, this function sets a pointer to a block of user data. This pointer can be recovered by a call to *VdrGetMovieUserData*.

**Parameters:**

*movieHandle:* The movie handle returned by *VdrAttachMovie*, *VdrAttachMovieWithMarks*, *VdrAttachOpenMovie*, or *VdrAttachOpenMovieWithMarks*.

*userData:* The pointer to a block of user data.

**Return Value:** TRUE if successful; otherwise, FALSE.

## VdrSetMpegChromaFormat

**#include "tekvdr.h"**

**BOOL VdrSetMpegChromaFormat**(**ResourceHandle** *resourceHandle*,
**MpegChromaFmt** *mpegChromaFmt*)

Set the chroma format used to encode video on the given resource handle.

**Parameters:**

*resourceHandle:* The handle returned by *VdrAllocateResource*.

*mpegChromaFmt:* The chroma format for the MPEG encoder resource, MpegChroma420 or MpegChroma422.

**Return Value:** TRUE if the call succeeds, FALSE otherwise.

**See Also:** *VdrGetMpegChromaFormat* on page 364.

## VdrSetMpegGopStructure

**#include "tekvdr.h"**

**BOOL VdrSetMpegGopStructure**(**ResourceHandle** *resourceHandle,*
**MpegGopEnd** *mpegGopEnd,*
**UINT** *reserved1,*
**UINT** *pPixPerIPix,*
**UINT** *bPixPerIPPix,*
**UINT** *reserved2*)

Set the group of pictures (GOP) structure used to encode video on the given resource handle.

**Parameters:**

| | |
|---|---|
| *resourceHandle:* | The handle returned by *VdrAllocateResource*. |
| *mpegGopEnd:* | Type of GOP end, GopOpenEnd or GopClosedEnd. |
| *reserved1:* | For compatibility with future code, this value must be set to `1`. |
| *pPixPerIPix:* | Number of P-frames generated per I-frame. A value of `0` will cause no P-frames to be encoded. |
| *bPixPerIPPix:* | Number of B-frames generated per anchor picture (I-frame or P-frame). A value of `0` will cause no B-frames to be encoded. |
| *reserved2:* | For compatibility with future code, this value must be set to `3`. |

**Return Value:** TRUE if the call succeeds, FALSE otherwise.

**See Also:** *VdrGetMpegGopStructure* on page 366.

## VdrSetPlayMode

**#include "tekvdr.h"**

**BOOL VdrSetPlayMode(VdrHandle** *vdrHandle,*
**MediaPlayMode** *mediaPlayMode, ...*)

This function causes motion play to occur in the given mode. The play mode is volatile, with a default of normal.

For PlaySlaved, the call is:

**BOOL VdrSetPlayMode(VdrHandle** *vdrHandle,*
**MediaPlayMode** *mediaPlayMode, VdrHandle masterport,*
*INT fieldOffset***)**

**When calling with PlaySlaved as *mediaPlayMode*, the position of the port represented by *vdrHandle* is slaved to the position of the port represented by *masterPort*, offset by the given *fieldOffset*. A positive *fieldOffset* causes the position of the port represented by *vdrHandle* to be greater than that of the port represented by *masterPort* by *fieldOffset* fields.**

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*mediaPlayMode:* One of the following:

- PlayNormal
- PlayLoop
- PlayBounce
- PlayLimited
- PlaySlaved

**Return Value:** TRUE if the playback mode was set; otherwise, FALSE.

**Comments:** The media limits are defined to be:

The smallest numbered field referenced by an in-point of any attached media, and the largest numbered field reference by an out-point of any attached media.

The clock limits are defined to be the field number set by *VdrSetMinPosition* and the field number before the field number set by *VdrSetMaxPosition*. The default value for either clock limit is `0`. The duration of a play with neither clock limit set is `0`. Clock limits are reset to default values by the *VdrResetPort* command.

Play always starts at the current position. If play is in the forward direction, it will hit the field of the larger limit first. If play is in the backward (negative) direction, it will hit the field of the smaller limit first.

**PlayNormal** causes play to ***stop at a media limit***.

**PlayLimited** causes play to ***stop at a clock limit***.

**PlayLoop** causes play to ***continue from the opposite clock limit*** repeatedly when it reaches a clock limit.

**PlayBounce** causes play to ***reverse direction upon reaching a clock limit***.

## VdrSetPosition

**#include "tekvdr.h"**

**BOOL VdrSetPosition(VdrHandle** *vdrHandle*,
**INT** *fieldNumber*)

This function causes the current position to be set to the given reference field number.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |
| *fieldNumber:* | The number of the field to which the current position should be set. |

**Return Value:** TRUE if the position was set to the given field; otherwise, FALSE.

## VdrSetPositionLimits

**#include "tekvdr.h"**

**BOOL VdrSetPositionLimits(VdrHandle** *vdrHandle*,
**INT** *minPosition*,
**INT** *maxPosition*)

This function has the same effect as separate calls to *VdrSetMinPosition* and *VdrSetMaxPosition*.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |
| *minPosition:* | The field position that will be the smallest clock value in **MediaPlayModes** other than **PlayNormal**. |
| *maxPosition:* | The field position that will be the largest clock value in **MediaPlayModes** other than **PlayNormal**. |

**Return Value:** TRUE if the limits were set; otherwise, FALSE.

## VdrSetPositionRelative

**#include "tekvdr.h"**

**BOOL VdrSetPositionRelative**(**VdrHandle** *vdrHandle*,
**INT** *fieldOffset*,
**VdrHandle** *masterPort*)

This function causes the current position to be set relative to the *masterPort* position.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. This is the port on which the actual positioning will occur. |
| *fieldOffset:* | The field data (+ or -) from the current position of the *masterPort*. |
| *masterPort*: | The port returned by *VdrOpenPortConnection* of the channel to which the *fieldNumber* is relative. |

**Return Value:** TRUE if the position is set correctly; otherwise, FALSE.

## VdrSetResourceUserData

**#include "tekvdr.h"**

**BOOL VdrSetResourceUserData**(**ResourceHandle** *resourceHandle*,
**void*** *userData*)

For the given resource handle, this function sets a pointer to a block of user data. This pointer can be recovered by a call to *VdrGetResourceUserData*.

**Parameters:**

| | |
|---|---|
| *resourceHandle:* | The handle returned by *VdrAllocateResource*. |
| *userData:* | The pointer to a block of user data. |

**Return Value:** TRUE if the pointer was set; otherwise, FALSE.

## VdrSetSavedVbiLines

**#include "tekvdr.h"**

**BOOL VdrSetSavedVbiLines**(**ResourceHandle** *resourceHandle*,
**UINT** *numLumaLines*,
**UINT*** *lumaLinesToSave*,
**UINT** *numChromaLines*,
**UINT*** *chromaLinesToSave*)

This function sets the numbers of the VBI lines to be losslessly recorded when recording MPEG video.

**Parameters:**

*resourceHandle:* handle to MPEG encoder, from *VdrAllocateResource*. For future compatibility, this function will accept handles to media encoders which will also provide this functionality.

*numLumaLines:* number of lines of luminance data to be saved, equivalent to the number of entries in lumaLinesToSave.

*lumaLinesToSave:* an array containing the numbers of the video lines of luminance data to be saved. The numbering follows the conventions in the CCIR-601 specification.

*numChromaLines:* number of lines of chrominance data to be saved, equivalent to the number of entries in chromaLinesToSave.

*chromaLinesToSave:* an array containing the numbers of the video lines of chrominance data to be saved.

The numbering follows the conventions in the CCIR-601 specification.

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** If numLumaLines and numChromaLines are 0, no VBI lines will be saved losslessly; this is the default value. If (numLumaLines + numChromaLines) exceeds 3 for PDR series Profile systems, or 6 for PVS Profile XP systems, the function will return false, and the VBI lines saved will not be affected by the function call. Compression of video begins on a line after the highest-numbered losslessly-recorded VBI line.

**See Also:** *VdrGetSavedVbiLines* on page 392.

## VdrSetStillPlayMode

**#include "tekvdr.h"**

**BOOL VdrSetStillPlayMode(VdrHandle** *vdrHandle*,
**StillMode** *stillPlayMode*)

This function sets the playback mode to be used during still play. When **PlayByField** is selected, playback occurs by line-doubling the current field. This will reduce motion effects at the expense of vertical resolution. When **PlayByFrame** is selected, playback occurs by playing the current and next field in sequence; this will enhance vertical resolution at the possible expense of added motion effects. This function has no effect on record mode.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*stillPlayMode:* One of the following:

- PlayByField
- PlayByFrame

**Return Value:** TRUE if the still play mode was successfully set to the given value; otherwise, FALSE.

## VdrSetVideoCodecTiming

**#include "tekvdr.h"**

**Comments:** New command; engineering specification not yet available.

## VdrSetVideoFormat

**#include "tekvdr.h"**

**BOOL VdrSetVideoFormat(VdrHandle** *vdrHandle*,
**VideoFormat** *videoFormat*)

This function sets the video format of the given Vdr port.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*videoFormat:* One of the following:

- Format525_60Hz_2To1
- Format625_50Hz_2To1
- Format720_59_94Hz_1To1
- Format720_60Hz_1To1
- Format1080_24Hz_1To1
- Format1080_25Hz_2To1
- Format1080_29_97Hz_2To1
- Format1080_30Hz_2To1

All 720 and 1080 formats are SMPTE 292 video formats.

**Return Value:** TRUE if successful; otherwise, FALSE.

## VdrSetVideoGoalSize

**#include "tekvdr.h"**

**BOOL VdrSetVideoGoalSize**(**ResourceHandle** *resourceHandle*,
**UINT** *bytesPerField*)

This function sets the goal size for video compression (applies to recording only) for the given video codec resource. This affects all video media files attached to this resource. Volatile, default is **75000**.

**Parameters:**

*resourceHandle:* The handle to the video codec resource returned by *VdrAllocateResource*.

*bytesPerField:* The goal size in bytes per field (unsigned integer).

**Return Value:** TRUE if the video goal size was set; otherwise, FALSE.

## VdrShuttle

**#include "tekvdr.h"**

**BOOL VdrShuttle(VdrHandle** *vdrHandle*,
**double** *shuttleRate*)

This function causes motion playback or recording. If already playing or recording in motion mode, changes the playback or record rate. If *VdrCuePlay* has been previously issued, playback will begin at the start of the next external reference vertical interval. If *VdrCueRecord* has been previously issued, recording will begin at the start of the next external reference vertical interval. If issued from an idle state, the latency of start of playback or record is not guaranteed.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |
| *shuttleRate:* | [-50.0 ... 50.0] A shuttle rate of 1.0 represents normal forward play or record. |

**Return Value:** TRUE if the playback or recording was set to shuttle; otherwise, FALSE.

## VdrShuttleAtGenTc

**#include "tekvdr.h"**

**BOOL VdrShuttleAtGenTc(VdrHandle** *vdrHandle*,
**double** *shuttleRate*,
**ResourceHandle** *genTcHandle*,
**UINT** *time*)

This function specifies a shuttle rate that will be scheduled to take effect at a specific time. This function is generally used to change the playback from 0.0 to 1.0 to start playback and to change the playback from 1.0 to 0.0 to end playback.

**Parameters:**

*vdrHandle*: The handle returned by *VdrOpenPortConnection*.

*shuttleRate*: [-50.0 … 50.0] A shuttle rate of 1.0 represents normal forward play or record.

*genTcHandle*: The handle returned by *VdrAllocateResource* for a timecode generator that will supply the reference time.

*time*: The time, encoded as timecode, at which the shuttle is to be scheduled. (See the SET_TIMECODE macro in *vdrtypes.h*)

**Return Value:** TRUE if successful; otherwise, FALSE;

**Comments:** The port must not be in an Idle state when this function is called. Calling *VdrShuttle* or *VdrIdle* for a port will cancel any outstanding *VdrShuttleAtGenTc* controls that may have been set for that port.

The time parameter must be one field ahead of the desired start time in order to avoid drop-frame issues and to maintain compatibility with the mechanism used by *VdrShuttle*.

For example, if the desired start time is 1:15:00:00, the specified time parameter must be 1:14:59:59 (the last two digits specify the field number). If drop-frame timecode is in use, and the desired start time is 10:00:00:04, the specified time parameter must be 9:59:59:59, taking into account the four dropped timecode fields.

This command must be issued no later than the latency value specified by *VdrGetStateChangeLatency* plus one field before the desired start time.

For example, if the latency value is 14 fields and the desired start time is 1:15:00:00, the command must be issued no later than 1:14:59:45. If drop-frame timecode is in use, the latency value is 14 fields, and the desired start time is 10:00:00:04, the command must be issued no later than 9:59:59:45, taking into account the four dropped timecode fields.

**See Also:** *VdrCancelAllPendingGpiTriggerCommands* on page 297,
*VdrGetStateChangeLatency* on page 393
*VdrSetGenTcRegenSource* on page 428,
*VdrSetGenTcFormat* on page 426,
*VdrSetGenTcMode* on page 427,
*VdrSetDefaultTcFormat* on page 421.

## VdrShuttleOnGpiInputTrigger

**#include "tekvdr.h"**

**BOOL VdrShuttleOnGpiInputTrigger(VdrHandle** *vdrHandle*,
**double** *shuttleRate*,
**ResourceHandle** *gpiInputResource*);

This function initiates a shuttle operation at a specified rate upon receipt of a GPI trigger on a specified input.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*shuttleRate:* [-50.0 ... 50.0] A shuttle rate of 1.0 represents normal forward play or record.

*gpiInputResource:* The handle to the resource of type **ResourceGpiInput** returned by *VdrAllocateResource*.

**Return Value:** TRUE if successful; otherwise, FALSE.

## VdrStateEvent

**#include "tekvdr.h"**

**BOOL VdrStateEvent(VdrHandle** *vdrHandle,*
**INT** *states,*
**EventType** *type, ...*)

This function specifies an event which will occur each time the port changes state into one of the selected states. The selected states are specified in the *states* parameter, and may be any combination of EventStateIdleRecord, EventStatePlayReady, and EventStatePlayActive. To make specification easier, the file *vdrtypes.h* defines the bitmasks for these states and for other combination states.

IdleRecord will be the active state of the selected port whenever the port is in idle or any record state. PlayReady will be the active state of the selected port whenever the port is in play-cue state, or the current position of the port playback can not advance because the port is in PlayLimited mode, and the PositionLimit has been reached. PlayActive will be the selected port's active state whenever the port is in play-shuttle, or play-jog, and is not limited by a PositionLimit.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |
| *state:* | The states in which this event should be active. |
| *type:* | Specific event type and supporting parameters. (See *Table 21* on page 305). |

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** The types of events listed in *Table 21* on page 305 can be used.

This function should be used instead of *VdrScheduleEvent* for playback connections that are coded to occur at the port timeline position of minus infinity (-veryLargeNumber).

**See Also:** *VdrDefaultEvent* on page 304,
*VdrScheduleEvent* on page 413.

## VdrStateEventAtGenTc

**#include "tekvdr.h"**

**BOOL VdrStateEventAtGenTc(VdrHandle** *vdrHandle*,
**INT** *stateMask*,
**ResourceHandle** *genTcHandle*,
**UINT** *time*,
**EventType** *type*, ...)

This function specifies an event that will be scheduled to take effect at the specified time, relative to the time in the timecode generator. After taking effect, the event will be active only when the port is in the selected states.

**Parameters:**

| | |
|---|---|
| *vdrHandle:* | The handle returned by *VdrOpenPortConnection*. |
| *stateMask:* | The states during which the event is to be active. |
| *genTcHandle:* | The handle returned by *VdrAllocateResource* for a timecode generator which is to supply the reference time. |
| *time:* | A time value at which the event should be scheduled. The time is relative to the value in the timecode generator. |
| *type:* | The specific event type and supporting parameters. (See *Table 21* on page 305.) |

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** The types of events listed in *Table 21* on page 305 can be used. The function returns immediately to the calling program. A successful return means that the event has been scheduled.

The time parameter must be one field ahead of the desired time in order to avoid drop-frame issues and to maintain compatibility with the mechanism used by *VdrShuttle*.

For example, if the desired time is 1:15:00:00, the specified time parameter must be 1:14:59:59 (the last two digits specify the field number). If drop-frame timecode is in use, and the desired time is 10:00:00:04, the specified time parameter must be 9:59:59:59, taking into account the four dropped timecode fields.

This command must be issued no later than the latency value specified by *VdrGetStateChangeLatency* plus one field before the desired start time.

For example, if the latency value is 14 fields and the desired time is 1:15:00:00, the command must be issued no later than 1:14:59:45. If drop-frame timecode is in use, the latency value is 14 fields, and the desired time is 10:00:00:04, the command must be issued no later than 9:59:59:45, taking into account the four dropped timecode fields.

**See Also:** *VdrSetDefaultTcFormat* on page 421,
*VdrSetGenTcRegenSource* on page 428,
*VdrSetGenTcFormat* on page 426,
*VdrSetGenTcMode* on page 427,
*VdrShuttleAtGenTc* on page 460.

## VdrStateEventOnGpiInputTrigger

**#include "tekvdr.h"**

**BOOL VdrStateEventOnGpiInputTrigger(VdrHandle** *vdrHandle*,
**INT** *stateMask*,
**ResourceHandle** *gpiInputResource*,
**EventType** *eventType*, ...);

This function specifies an event that will be scheduled to take effect upon receipt of a GPI trigger on the specified input. After taking effect, the event will be active only when the port is in the selected states.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*stateMask:* The states during which the event is to be active.

*gpiInputResource:* The handle to the resource of type **ResourceGpiInput** returned by *VdrAllocateResource*.

*eventType:* The specific event type and supporting parameters. (See *Table 21* on page 305.)

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** The types of events listed in *Table 21* on page 305 can be used. The function returns immediately to the calling program. A successful return means that the event has been scheduled.

**See Also:** *VdrSetDefaultTcFormat* on page 421,
*VdrSetGenTcRegenSource* on page 428,
*VdrSetGenTcFormat* on page 426,
*VdrSetGenTcMode* on page 427,
*VdrShuttleAtGenTc* on page 460.

## VdrTriggerGpiOutput

**#include "tekvdr.h"**

**BOOL VdrTriggerGpiOutput(ResourceHandle** *gpiOutputResource*);

This function triggers the specified GPI output to the active state.

**Parameters:**

*gpiOutputResource:* The handle to the resource of type **ResourceGpiOutput** returned by *VdrAllocateResource*.

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** Active state high or low and pulse width are set in the Profile XP Configuration Manager. The specified output remains active for the number of fields specified as the pulse width for the output.

## VdrTriggerGpiOutputOnGpiInputTrigger

**#include "tekvdr.h"**

**BOOL VdrTriggerGpiOutputOnGpiInputTrigger(ResourceHandle** *gpiOutputResource*, **ResourceHandle** *gpiInputResource*);

This function triggers the specified GPI output to the active state upon receipt of a GPI trigger on the specified input.

**Parameters:**

*gpiOutputResource:* The handle to the resource of type **ResourceGpiOutput** returned by *VdrAllocateResource*.

*gpiInputResource:* The handle to the resource of type **ResourceGpiInput** returned by *VdrAllocateResource*.

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** Active state high or low and pulse width are set in the Profile XP Configuration Manager. The specified output remains active for the number of fields specified as the pulse width for the output.

## VdrTriggerGpiOutputAtGenTc

**#include "tekvdr.h"**

**BOOL VdrTriggerGpiOutputAtGenTc(ResourceHandle** gpiOutputResource,
**ResourceHandle** genTcHandle,
**UINT** time);

This function triggers the specified GPI output to the active state at the specified time, relative to the time in the timecode generator.

**Parameters:**

*gpiOutputResource:* The handle to the resource of type **ResourceGpiOutput** returned by *VdrAllocateResource*.

*genTcHandle:* The handle returned by *VdrAllocateResource* for a timecode generator which is to supply the reference time.

*time:* A time value at which the GPI trigger should be scheduled. The time is relative to the value in the timecode generator.

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** Active state high or low and pulse width are set in the Profile XP Configuration Manager. The specified output remains active for the number of fields specified as the pulse width for the output.

## VdrWaitForGpiInputTrigger

**#include "tekvdr.h"**

**BOOL VdrWaitForGpiInputTrigger(VdrHandle** *vdrHandle*,
**ResourceHandle** *gpiInputResource*,
**BOOL *** *isActive*,
**DWORD** *dwMilliseconds*);

This function suspends the thread until the specified GPI input is triggered, or until the optional timeout period elapses.

**Parameters:**

*vdrHandle:* The handle returned by *VdrOpenPortConnection*.

*gpiInputResource:* The handle to the resource of type **ResourceGpiInput** returned by *VdrAllocateResource*.

*isActive:* A non-NULL pointer to the value representing the GPI input state. TRUE if active; otherwise, FALSE. This value is set when the function returns.

*dwMilliseconds:* The number of milliseconds until the wait times out. If *dwMilliseconds* is 0, the function returns immediately. If INFINITE, the function waits without timing out until the GPI input is triggered. (INFINITE is a macro defined in *winbase.h*.)

**Return Value:** TRUE if the function succeeds; otherwise, FALSE. Failure may be due to invalid parameters or an internal failure.

**Comments:** The GPI input is read every frame at the start of Field 1.

Use the return value and the value in *isActive* to distinguish between a return because of a GPI trigger, or a return because of a failure or timeout.

Setting *dwMilliseconds* to 0 is a simple way to poll a GPI input.

## VdrWaitForVideoField

**#include "tekvdr.h"**

**BOOL VdrWaitForVideoField(ConnectHandle** *connHandle*,
**VideoFieldType** *videoFieldType*)

This function will block the current thread until an interrupt for the given video field type occurs.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*videoFieldType:* The field type to the video returned by one of the following values used to specify the type of video field in a field sequence [0 ..7]:

- VideoField0246 (fields 0, 2, 4, 6)
- VideoField1357 (fields 1, 3, 5, 7)
- VideoField04 (fields 0, 4)
- VideoField0 (field 0)

**Return Value:** TRUE if the interrupt occurs; FALSE if a timeout occurs.

## VdrWhenIsEvent

**#include "tekvdr.h"**

**INT VdrWhenIsEvent**(**EventHandle** *eventHandle*)

This function returns the video field number at which the given event is scheduled.

**Parameters:**

*eventHandle:* The handle to the event returned by **ScheduleEvent**.

**Return Value:** This function returns the video field at which event is scheduled.

**Comments:** The following table shows the embedded disk recorder video subsystem state transitions in response to transport control commands.

# The Vfs library

The TekVfs library allows a low level interface to the embedded system's media files.

## VfsCancelCopy

**#include "tekvfs.h"**

**BOOL VfsCancelCopy**(**Handle** *operationHandle*)

This function cancels the copy which was started with *VfsCopyFile*.

**Parameters:**

*operationHandle:* The handle to the operation returned by *VfsCopyFile*.

**Return Value:** TRUE if the copy was canceled; otherwise, FALSE.

## VfsCloseHandle

**#include "tekvfs.h"**

**BOOL VfsCloseHandle(HANDLE** *fileHandle*)

This function closes the given file handle.

**Parameters:** See CloseHandle in the Win32_SDK for details.

**Return Value:** TRUE if successful; otherwise, FALSE.

## VfsCopyFile

**#include "tekvfs.h"**

**HANDLE VfsCopyFile(ConnectHandle** *connHandle*,
**const CHAR*** *sourceFile*,
**const CHAR*** *destFile*,
**BOOL** *waitForCompletion*)

This function copies the sourceFile to the destFile, either sychronously or asychronously. If the destination file exists and is Read-Only, then the copy will fail.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*sourceFile:* The name of the file to be copied.

*destFile*: The name of the file to be created.

*waitForCompletion*:If this is zero (FALSE) and copy initialization is successful, the copy will be performed asynchronously and control will be returned to the calling thread. In this case the caller must call *VfsStatusOfCopy* to wait for completion of the copy operation or cancel the copy operation with *VfsCancelCopy*. If this parameter is non-zero (TRUE), the copy will be performed synchronously and control will not be returned until the copy is complete.

**Return Value:** The returned handle can be used in subsequent calls to *VfsStatusOfCopy* and *VfsCancelCopy*. If *waitForCompletion* was set, the returned handle will be `0` for successful completion. If the copy operation fails, the function returns `INVALID_HANDLE_VALUE`.

## VfsCopyMediaSegment

**#include "tekvdr.h"**

**HANDLE VfsCopyMediaSegment**(**ConnectHandle** *connHandle,*
**const CHAR*** *sourceFile,*
**const CHAR*** *destFile,*
**ULONG** *sourceFirstField,*
**ULONG** *sourceLastField,*
**ULONG** *destFirstField,*
**BOOL** *waitForCompletion*)

This function copies the *sourceFile* to the *destFile*, either synchronously or asynchronously. If the destination file exists and is Read-Only, the copy will fail; if the source file is MPEG, the copy will return an error.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*sourceFile:* The name of the file to be copied.

*destFile*: The name of the file to be created.

*sourceFirstField*: The first field in the source file to include in the copied media segment.

*sourceLastField:* The field after the last field in the source file which will be included in the copied media segment; *sourceLastField* is not included in the copy.

*destFirstField*: The field in the destination file at which to begin playing the copied media segment.

*waitForCompletion*:If this is zero (FALSE) and copy initialization is successful, the copy will be performed asynchronously and control will be returned to the calling thread. In this case the caller must call *VfsStatusOfCopy* to wait for completion of the copy operation or cancel the copy operation with *VfsCancelCopy*. If this parameter is non-zero (TRUE), the copy will be performed synchronously and control will not be returned until the copy is complete.

**Return Value:** The returned handle can be used in subsequent calls to *VfsStatusOfCopy* and *VfsCancelCopy*. If *waitForCompletion* was set, the returned handle will be `0` for successful completion. If the copy operation fails, the function returns `INVALID_HANDLE_VALUE`.

## VfsCreateDirectory

**#include "tekvfs.h"**

**BOOL VfsCreateDirectory**(**ConnectHandle** *connHandle*,
**CHAR*** *dirPath*)

This function creates a directory.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *dirPath:* | The name of the directory to be created. |

**Return Value:** TRUE if the directory was created; otherwise, FALSE.

## VfsCreateFile

**#include "tekvfs.h"**

**HANDLE VfsCreateFile(ConnectHandle** *connHandle,*
**const CHAR*** *fileName,*
**DWORD** *access,*
**DWORD** *shareMode,*
**DWORD** *create,*
**DWORD** *attribs,*
**HANDLE** *templateFile*)

This function is the equivalent of the Windows NT CreateFile functions. It opens the specified file, creating the file if required.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

(See also **CreateFile** in the Win32_SDK for details.)

**Return Value:** This function returns a handle to the open file.

## VfsCreateMediaFile

**#include "tekvfs.h"**

**BOOL VfsCreateMediaFile(ConnectHandle** *connHandle,*
**CHAR*** *szFilename,*
**FileType** *nFiletype,*
**VideoFormat** *videoFormat*)

This function creates a media file of the given type and format.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*szFilename:* The name of the file to create.

*nFiletype:* One of the following:

- FileTypeJpegVideo (same as FileTypeVideo)
- FileTypeMpegVideo
- FileTypeMpeHiDefVideo
- FileTypeDvcpro25Video
- FileTypeDvcpro50Video
- FileTypeAudio
- FileTypeTimecode
- FileTypeNonMedia

*videoFormat:* One of the following:

- Format525_60Hz_2To1
- Format625_50Hz_2To1
- Format720_59_94Hz_1To1
- Format720_60Hz_1To1
- Format1080_24Hz_1To1
- Format1080_25Hz_2To1
- Format1080_29_97Hz_2To1
- Format1080_30Hz_2To1

All 720 and 1080 formats are SMPTE 292 video formats.

**Return Value:** TRUE if the file was created; otherwise, FALSE.

**Comments:** This function fills a small hole where a programmer is using *VfsFieldRead*, and then *VfsFieldWrite* to copy the contents of one media file to a new one. Otherwise, to create a new media file it must be attached to the timeline with the appropriate resource before it can be written to. With the inclusion of this new function, the manipulation of the timeline is no longer necessary.

**See Also:** *VfsCreateFile* on page 476, *VfsFieldRead* on page 482, and *VfsFieldWrite* on page 483.

## VfsDeleteFile

**#include "tekvfs.h"**

**BOOL VfsDeleteFile**(**ConnectHandle** *connHandle*,
**CHAR *** *fileName*)

This function deletes an existing file or directory (including subdirectories) in the embedded disk recorder file system.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *fileName:* | The null-terminated name of file to be deleted. |

**Return Value:** TRUE if the file was deleted; otherwise, FALSE.

## VfsDirectoryExists

**#include "tekvfs.h"**

**BOOL VfsDirectoryExists(ConnectHandle** *connHandle*,
**const CHAR*** *directoryName*)

This function returns TRUE if the given directory exists, FALSE otherwise.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.
*directoryName:* The name of the directory.

**Return Value:** TRUE if the directory exists; otherwise, FALSE.

## VfsEraseMediaSegment

**#include "tekvfs.h"**

**BOOL VfsEraseMediaSegment**(**ConnectHandle** *connHandle*,
**const CHAR*** *mediaFilePath*,
**INT** *inField*,
**INT** *outField*)

This function erases the given segment within a media file. For video media files, the erased segment will appear as black during playback. For audio media files, the erased segment will be silent during playback. All media handles for the given media file are affected by this operation. This command frees up disk space.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *mediaFilePath:* | The path to the media file. |
| *inField:* | The first field of deleted segment. |
| *outField:* | The field following last field of deleted segment. |

**Return Value:** TRUE if the segment was deleted; otherwise, FALSE.

## VfsFieldFree

**#include "tekvfs.h"**

**void VfsFieldFree**(**ConnectHandle** *connHandle*,
**UCHAR*** *memPointer*)

This function frees the memory allocated by the routine *VfsFieldRead*.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *memPointer:* | The pointer to the memory to free. |

**Return Value:** None (void function).

**Comments:** This function is necessary due to changes in the way Microsoft Visual C++ version 4.0 allocates/frees memory. Version 4.0 applications will fault if they attempt to use this function to deallocate memory returned by *VfsFieldRead*.

**See Also:** *VfsFieldRead* on page 482.

## VfsFieldRead

**#include "tekvfs.h"**

**unsigned char *VfsFieldRead(HANDLE** *fileHandle*,
**INT** *fieldNumber*,
**DWORD*** *dataSize*)

This function reads a field of data from the given media file.

**Parameters:**

| | |
|---|---|
| *fileHandle:* | The file handle from *VfsCreateFile*. |
| *fieldNumber:* | The number of the field to read. |
| *dataSize:* | Returns the useful size of the data in the buffer. |

**Return Value:** If successful, a malloc'ed buffer is returned. The useful size of the data in the buffer is returned in **dataSize*.

For timecode and audio files, the buffer will contain the raw data as stored internally. Fields from JPEG video media files are translated into a generically-readable JPEG data stream, which is JFIF-conformant. Fields from MPEG video media files contain the MPEG picture at the specified field and all of its associated MPEG bitstream headers.

The buffer returned must be freed by *VfsFieldFree*.

**Comments:** At this time, error reporting is all but non-existent. This function either works or fails.

**See Also:** *VfsFieldFree* on page 481,
*VfsFieldWrite* on page 483.

## VfsFieldWrite

**#include "tekvfs.h"**

**BOOL VfsFieldWrite(HANDLE** *fileHandle,*
**INT** *fieldNumber,*
**CONST** *void* buffer,*
**UINT** *bufferSize,*
**DWORD** **bytesWritten*)

This function writes a field of data to the given media file.

**Parameters:**

| | |
|---|---|
| *fileHandle:* | The path to a file in the embedded disk recorder file system. |
| *fieldNumber:* | The number of the field to write. |
| *buffer:* | The pointer to the buffer holding data to be written. |
| *bufferSize:* | Filled with the number of bytes consumed. |

**Return Value:** If successful, this function returns a non-zero value and **bytesWritten* is filled with how many bytes were consumed. This should match *bufferSize*.

**Comments:** For timecode and audio files, the buffer must contain raw data as stored internally, of the correct size. For fields written to video media files, the input must be a JPEG data-stream with these constraints:

| | |
|---|---|
| H | The Huffman tables must match those used internally. (These are not currently checked, but should induce an error). |
| H | The coding must be three components: Y, Cb, Cr. |
| H | The sampling ratio of these components (part of the frame signal) must be: 2x1, 1x1, 1x1. |

There will probably be additional constraints, which will be removed for some field writing method, not yet determined.

At this time, error reporting is all but non-existent. This function either works or fails.

## VfsFileExists

**#include "tekvfs.h"**

**BOOL VfsFileExists(ConnectHandle** *connHandle*,
**const CHAR*** *fileName*)

This function returns TRUE if the given file exists, FALSE otherwise.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *fileName:* | The name of the file. |

**Return Value:** TRUE if the file exists; otherwise, FALSE.

## VfsFindClose

**#include "tekvfs.h"**

**BOOL VfsFindClose(HANDLE** *findHandle*)

This function closes the given search handle.

**Parameters:**

*findHandle:* The handle returned by *VfsFindFirstFile*.

**Return Value:** TRUE if the handle was closed; otherwise, FALSE.

## VfsFindFirstFile

**#include "tekvfs.h"**

**HANDLE VfsFindFirstFile(ConnectHandle** *connHandle*,
**const CHAR*** *searchFile*,
**WIN32_FIND_DATA*** *findData*)

This function searches a directory for a file whose name matches the specified filename.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *searchFile:* | The null-terminated string specifying a valid directory or path and filename. It can include ***** for wild-card matching of either prefix or suffix. |
| *findData:* | Points to the structure which receives information about the found file. See **FindFirstFile** in the Win32_SDK for details. |

**Return Value:** If the search succeeds, the function returns a search handle for use with *VfsFindNextFile* and *VfsFindClose*; it returns `INVALID_HANDLE_VALUE` otherwise.

## VfsFindNextFile

**#include "tekvfs.h"**

**BOOL VfsFindNextFile(HANDLE** *findHandle*,
**WIN32_FIND_DATA*** *findData*)

This function continues a file search from a previous call to *VfsFindFirstFile*.

**Parameters:**

| | |
|---|---|
| *findHandle:* | The handle returned by *VfsFindFirstFile*. |
| *findData:* | Points to the structure which receives information about the found file. See **FindFirstFile** in the Win32_SDK for details. |

**Return Value:** TRUE if the search succeeds; otherwise, FALSE.

## VfsGetCurrentDirectory

**#include "tekvfs.h"**

**DWORD VfsGetCurrentDirectory**(**ConnectHandle** *connHandle*,
**DWORD** *bufferSize*,
**CHAR*** *currentDirectory*)

This function returns the current directory for this process.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *bufferSize:* | The size of *currentDirectory*; must include space for the null character. |
| *currentDirectory:* | Points to a buffer to hold null-terminated current directory string. |

**Return Value:** If successful, the function returns the number of characters written to the *currentDirectory* buffer, not including the null character; `0` otherwise.

## VfsGetCurrentDrive

**#include "tekvfs.h"**

**UINT VfsGetCurrentDrive(ConnectHandle** *connHandle*)

This function returns the current logical group set.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the current logical group set.

## VfsGetEngineMajorVersion

**#include "tekvfs.h"**

**UINT VfsGetEngineMajorVersion**(**ConnectHandle** *connHandle*)

This function returns the major version number of the embedded system engine. For example, the major number in version 1.23 is **1**.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the major number of the embedded system engine.

## VfsGetEngineMinorVersion

**#include "tekvfs.h"**

**UINT VfsGetEngineMinorVersion(ConnectHandle** *connHandle*)

This function returns the minor version number of the embedded system engine. For example, the minor number in version 1.23 is **23**.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the major number of the embedded system engine.

## VfsGetFileAttributes

**#include "tekvfs.h"**

**DWORD VfsGetFileAttributes**(**ConnectHandle** *connHandle*,
**const CHAR*** *fileName*)

This function returns attributes for the given file.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*fileName:* Null-terminated name of the file.

**Return Value:** If successful, the function returns the attributes for the file; `xFFFFFFFF` if not. See GetFileAttributes in the Win32_SDK for details.

## VfsGetFileDefaultMarkIn

**#include "tekvfs.h"**

**UINT VfsGetFileDefaultMarkIn(ConnectHandle** *connHandle*,
**const CHAR*** *filePath*)

This function returns the default mark-in point for the given media file. This is equal to first recorded field in the file.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*filePath:* The path to the media file.

**Return Value:** This function returns the default mark-in point.

## VfsGetFileDefaultMarkOut

**#include "tekvfs.h"**

**UINT VfsGetFileDefaultMarkOut(ConnectHandle** *connHandle,*
**const CHAR*** *filePath*)

This function returns the default out-point for the given media file. This is equal to the field after the last recorded field in the file.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *filePath:* | Path to media file. |

**Return Value:** This function returns the default mark out-point.

## VfsGetFileDefaultMarks

**#include "tekvfs.h"**

**BOOL VfsGetFileDefaultMarks(ConnectHandle** *connHandle*,
**const CHAR*** *filePath*,
**UINT*** *defaultMarkIn*,
**UINT*** *defaultMarkOut*)

This function returns both the default in-point and default out-point in a single function call.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *filePath:* | The path to the media file being queried. |
| *defaultMarkIn:* | A pointer to a **UINT** into which the *defaultMarkIn* value can be stored. |
| *defaultMarkOut:* | A pointer to a **UINT** into which the *defaultMarkOut* value can be stored. |

**Return Value:** TRUE if the values are stored; otherwise, FALSE.

**Comments:** The *defaultMarkIn* of a media file is the first field position in the file with recorded material. The *defaultMarkOut* of a media file is the first field position beyond the last field of media file to contain recorded material.

## VfsGetFileModificationTime

**#include "tekvfs.h"**

**FILETIME VfsGetFileModificationTime**(**ConnectHandle** *connHandle*,
**const CHAR*** *fileName*)

This function returns the modification time for the given file.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.
*fileName:* The desired filename.

**Return Value:** This function returns the modification time for the given file in a FILETIME structure. See the Win32_SDK for details.

## VfsGetFileRecordedDuration

**#include "tekvfs.h"**

**UINT VfsGetFileRecordedDuration(ConnectHandle** *connHandle*,
**const CHAR*** *filePath*)

This function returns the number of fields between the first and last recorded field in the file. This is the same as *VfsGetFileDefaultMarkIn* and *VfsGetFileDefaultMarkOut*.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*filePath:* The path name to a media file in the embedded disk recorder file system.

**Return Value:** This function returns the difference between the field numbers of the first recorded field and the last recorded field of the file.

## VfsGetFileSize

**#include "tekvfs.h"**

**Comments:** New command; engineering specification not yet available.

## VfsGetFileTimeCode

**#include "tekvfs.h"**

**Comments:** New command; engineering specification not yet available.

## VfsGetFileType

**#include "tekvfs.h"**

**FileType VfsGetFileType**(**ConnectHandle** *connHandle*,
**const CHAR*** *filePath*)

This function returns the type of the given file in the embedded disk recorder file system.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*filePath:* The path to a file in the embedded disk recorder file system.

**Return Value:** This function returns one of the following:

- FileTypeJpegVideo (same as FileTypeVideo)
- FileTypeMpegVideo
- FileTypeMpeHiDefVideo
- FileTypeDvcpro25Video
- FileTypeDvcpro50Video
- FileTypeAudio
- FileTypeTimecode
- FileTypeNonMedia

## VfsGetFileVideoFormat

**#include "tekvfs.h"**

**VideoFormat VfsGetFileVideoFormat(ConnectHandle** *connHandle*,
**const CHAR*** *filePath*)

This function returns the video format of the given file in the embedded disk recorder file system.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*filePath:* The path to a file in the embedded disk recorder file system.

**Return Value:** This function returns the video format of the given file in the embedded disk recorder file system.

## VfsGetMajorVersion

**#include "tekvfs.h"**

**UINT VfsGetMajorVersion**(**ConnectHandle** *connHandle*)

This function returns the major version number of the TekVfs library. For example, the major number in version 1.23 is **1**.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the major number of the TekVfs library.

## VfsGetMinorVersion

**#include "tekvfs.h"**

**UINT VfsGetMinorVersion(ConnectHandle** *connHandle*)

This function returns the minor version number of the TekVfs library. For example, the minor number in version 1.23 is **23**.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the major number of the TekVfs library.

## VfsGetUsableSpace

**#include "tekvfs.h"**

**BOOL VfsGetUsableSpace(ConnectHandle** *connhandle,*
**INT** *fileSystemLogicalSetID***,**
**FileType** *encodingType***,**
**INT** *videoBitRate***,**
**INT** *audioSampleSize***,**
**ULARGE_INTEGER*** *usableSpace* **)**

This call returns the approximate number of bytes available for storing video/audio/timecode data in the location pointed to by the *usableSpace* parameter. The number returned does not need to be adjusted to account for file system overhead or padding. All reported usable space is available for data.

**Parameters:**

*connhandle*: The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*fileSystemLogicalSetID*:The ID of the disk set you want to check the free space on.

*encodingType:* The type of encoding used for the data to be transfered or recorded. FileType is from *vdrtypes.h*, and can have one of the following values for this call:

- FileTypeJpegVideo (same as FileTypeVideo)
- FileTypeMpegVideo
- FileTypeMpeHiDefVideo
- FileTypeDvcpro25Video
- FileTypeDvcpro50Video
- FileTypeAudio
- FileTypeTimecode
- FileTypeNonMedia

*videoBitRate:* The video rate in bits per second for the data you will be transfering. (bits per second at normal playback speed, not the transfer rate)

*audioSampleSize*: The number of bits per audio sample, 16 or 24.

*usableSpace* A pointer to the location the result should be stored in.

**Return Value:** TRUE if the usable space was obtained; otherwise, FALSE.

**Comments:** A small amount of space is consumed by clip overhead for each clip. This function includes clip overhead for only one clip when it calculates the usable space. If more than one clip is created, this function should be called periodically to reflect the space used for clip overhead. Calling this function after filling half the reported usable space provides more reliable estimates without excessive calculation.

When recording short clips, an encoder can exceed its goal rate by a significant amount. On clips of approximately 30 frames in length, the rate may be up to 25% over the goal rate. This extra data must be included in calcuations of time remaining based on the reported usable space.

## VfsMoveFile

**#include "tekvfs.h"**

**BOOL VfsMoveFile**(**ConnectHandle** *connHandle*,
**const CHAR*** *existingFile*,
**const CHAR*** *newFile*)

This function renames an existing file or directory, including subdirectories.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *existingFile:* | The null-terminated name of the existing file. |
| *newFile:* | The null-terminated new name of the file. |

**Return Value:** TRUE if the file was renamed; otherwise, FALSE.

## VfsMoveMediaSegment

**#include "tekvfs.h"**

**HANDLE VfsMoveMediaSegment**(**ConnectHandle** *connHandle*,
**const CHAR*** *sourceFile*,
**const CHAR*** *destFile*,
**ULONG** *sourceFirstField*,
**ULONG** *sourceLastField*,
**ULONG** *destFirstField*,
**BOOL** *waitForCompletion*)

This function moves the sourceFile to the destFile, either synchronously or asynchronously. If the destination file exists and is Read-Only, then the move will fail.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*sourceFile:* Name of the file containing the media segment to be moved.

*destFile*: The location of the file where the media segment is to moved.

*sourceFirstField*: The first field in the source file to move the media segment to.

*sourceLastField:* The last field in the source file to include the moved media segment.

*destFirstField*: The field in the destination file where the first moved field is placed.

*waitForCompletion*:If *this* is non-zero (TRUE), then the move will be performed synchronously and control will not be returned until the move is complete. If *waitForCompletion* is zero (FALSE) and the move starts successfully, it will be performed asynchronously and control will be returned to the calling thread. In this case, the caller must call *VfsStatusOfCopy* to wait for completion of the copy or cancel the copy with *VfsCancelCopy*.

**Return Value:** The returned handle can be used in subsequent calls to **VfsStatusOfMove** and **VfsCancelMove**. If the copy operation fails, the function returns `INVALID_HANDLE_VALUE`.

**Comments:** The source and destination files containing the affected media must be local to the machine on which the move is being performed (on the same machine *connHandle* points to).

## VfsQueryFileSystemSpace

**#include "tekvfs.h"**

**BOOL VfsQueryFileSystemSpace**(**ConnectHandle** *connHandle*,
**INT** *fileSystemLogicalSetId*,
**ULARGE_INTEGER** **totalSpace*,
**ULARGE_INTEGER** **spaceRemaining*)

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*FileSystemLogicalSetId:* The ID for one of the systems into which the media disk has been partitioned. This value must be less than the value returned by the function *CfgGetNumFileSystems*. File system 0 always exists, and the disks that are internal to the Profile video server will be assigned to that ID.

*totalSpace:* Total size (in bytes) of the specified file system.

*spaceRemaining:* Free space (in bytes) on the specified file system.

**Return Value:** TRUE if *totalSpace* and *spaceRemaining* are obtained without problems; otherwise, FALSE.

## VfsReadFile

**#include "tekvfs.h"**

**BOOL VfsReadFile(HANDLE** *fileHandle*,
**void** **buffer*,
**DWORD** *bytesToRead*,
**DWORD*** *bytesRead*)

This function reads from the given file handle in the embedded disk recorder system. See the call **ReadFile** in the Win32_SDK for details.

**Parameters:**

*fileHandle:* The file handle returned by *VfsCreateFile*.

**Return Value:** TRUE if successful; otherwise, FALSE.

## VfsSetCurrentDirectory

**#include "tekvfs.h"**

**BOOL VfsSetCurrentDirectory**(**ConnectHandle** *connHandle***,**
**const CHAR*** *currentDirectory*)

This function sets the current directory for this process.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.
*currentDirectory:* A null-terminated string specifying the desired directory.

**Return Value:** TRUE if successful; otherwise, FALSE.

## VfsSetFileAttributes

**#include "tekvfs.h"**

**BOOL VfsSetFileAttributes**(**ConnectHandle** *connHandle*,
**const CHAR*** *fileName*,
**DWORD** *fileAttributes*)

This function sets the attributes for the given file.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *fileName:* | The null-terminated name of the file. |
| *fileAttributes:* | See **SetFileAttributes** in the Win32_SDK for details. |

**Return Value:** TRUE if successful; otherwise, FALSE.

## VfsStatusOfCopy

**#include "tekvfs.h"**

**BOOL VfsStatusOfCopy**(**Handle** *operationHandle*,
**BOOL** *waitForCompletion*,
**UINT** **pctDone*,
**ULONG** **copyError*,
**UINT** **bytesLastSecond*,
**UINT** **bytesPerSecond*)

This function checks the status of a *VfsCopyFile* call for the given *operationHandle*. The call can be synchronous or asynchronous, depending on the value of *waitForCompletion*.

**Parameters:**

*operationHandle:* The handle returned by *VfsCopyFile*.

*waitForCompletion:* If this is zero (FALSE), then the call will not wait for the copy operation to complete. If the copy is complete, *pctDone* will return a value of `100`; if *pctDone* returns with any other value, then the copy is not completed and the caller can cancel the copy operation with *VfsCancelCopy* or call *VfsStatusOfCopy* again to wait for completion of the copy. If this parameter is non-zero (TRUE), the call will wait for the copy operation to complete and *pctDone* will return a value of `100`.

*pctDone:* A value of `100` indicates that the copy operation is completed; any other value indicates that the operation has not yet completed. A null value can be specified for *pctDone* if *waitForCompletion* is non-zero (TRUE).

*copyError:* This indicates any error that occurs during the copy. A value of zero indicates no errors. This field is only valid if *pctDone* returns `100`. If *copyError* is null, no error value is returned.

*bytesLastSecond:* If *bytesLastSecond* is non-null, then *bytesLastSecond* returns the copy rate for the last second of the copy.

*bytesPerSecond:* If *bytesPerSecond* is non-null, then *bytesPerSecond* returns the copy rate up until the point of the call to *VfsStatusOfCopy*.

**Return Value:** This function returns TRUE if the copy succeeds. (Success also depends on the values of *pctDone* and *copyError*.) It returns FALSE if the status request fails or if an invalid operationHandle is specified.

## VfsStripeTimecode

**#include "tekvfs.h"**

**BOOL VfsStripeTimecode**(**ConnectHandle** *connHandle*,
**const CHAR*** *filePath*,
**UINT** *startingTc*,
**UINT** *userBits*,
**TcFormat** *tcFormat*)

Stripes the timecode media file represented by filePath in the given video format and timecode format, beginning with the given timecode.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *filePath:* | Path to the timecode media file. |
| *startingTc:* | Starting timecode, as constructed by SET_TIMECODE. (See the SET_TIMECODE macro in *vdrtypes.h*) |
| *userBits:* | User bits to be striped throughout file; see EventSetGtcBits for format. |
| *tcFormat:* | Timecode format for the timecode file, one of the following:<br>• TcFormatDropFrame<br>• TcFormatNonDropFrame<br>(There is no effect if the video format of the given file is `Format625_50Hz_2To1`). |

**Returns:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** Fails on non-existent files or on files which are not timecode media files. Striping affects only fields which have not been recorded in a timecode media files, and will not replace recorded fields. To erase all previously-recorded fields, call *VfsEraseMediaSegment* before calling this function.

## VfsWriteFile

**#include "tekvfs.h"**

**BOOL VfsWriteFile(HANDLE** *fileHandle*,
**void*** *buffer*,
**DWORD** *bytesToWrite*,
**DWORD*** *bytesWritten*)

This function writes the given buffer to the given file. See the call **WriteFile** in the Win32_SDK for details.

**Parameters:**

*fileHandle:* The file handle from *VfsCreateFile*.

**Return Value:** TRUE if successful; otherwise, FALSE.

# The Vme library

The TekVme library allows control of the video mix effects board.

## VmeAllocateContext

**#include "tekvme.h"**

**VmeHandle VmeAllocateContext**(**ConnectHandle** *connHandle*,
**ResourceType** *type*)

This function gets a free mix context, initializes it, and returns a handle to it for use. The resource type must be specified so that the correct type of mix context is built.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*type:* Resource type for a mixer.

**Return Value:** This function returns a mix context handle if one is available; NULL otherwise.

**Comments:** The only valid resource type at this time is ResourceVideoCombiner.

## VmeApplyContext

**#include "tekvme.h"**

**BOOL VmeApplyContext**(**ResourceHandle** *resourceHandle*,
**VmeHandle** *vmeHandle*)

This function maps the mix context to the mix effects hardware represented by *resourceHandle*. The function will error if the type of the mix context does not match the type of the specified resource handle.

**Parameters:**

*resourceHandle:* The resource handle for a mixer returned by *VdrAllocateResource*.

*vmeHandle:* A mix context handle.

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeFreeContext

**#include "tekvme.h"**

**BOOL VmeFreeContext**(**VmeHandle** *vmeHandle*)

This function returns a mix context back to the system.

**Parameters:**

*vmeHandle:* A mix context handle.

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeGetAspect

**#include "tekvme.h"**

**INT VmeGetAspect(VmeHandle** *hMix*,
**VmeStage** *stage*)

This function returns the aspect ratio number for the transition set in the mix context specified by the given handle. The aspect ratio number is a fixed point value, with `1000` (defined by `VME_NORMAL_ASPECT`) being 1:1 (horizontal:vertical). Twice that value will give a 2:1 aspect, while half will result in 1:2.

**Parameters:**

| | |
|---|---|
| *hMix:* | Handle to the mix context allocated by *VmeAllocateContext*. |
| *stage:* | Value specifying a mixer stage. Must be `VmeFirstStage`. |
| *nAspect:* | The aspect ratio fixed point number. |

**Return Value:** Returns the aspect value currently in the specified mix context.

**Comments:** Only values greater than zero have any meaning, and some transitions may limit this value to certain ranges of aspect. For example, circles won't work outside the 2:1 or 1:2 range. This aspect value will have no effect on transitions that have no aspect (like horizontal wipes).

**See Also:** *VmeGetRotation* on page 531,
*VmeSetAspect* on page 538, and
*VmeSetRotation* on page 550.

## VmeGetBorderColor

**#include "tekvme.h"**

**BOOL VmeGetBorderColor**(**VmeHandle** *vmeHandle*,
**VmeStage** *stage*,
**INT*** *pnHue*,
**INT*** *pnChroma*,
**INT*** *pnLum*)

This function gets the color of the wipe transition border in the specified mixer stage.

**Parameters:**

| | |
|---|---|
| *vmeHandle:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**. |
| *pnHue:* | The pointer to the location for hue value, range [0 .. VME_MAX_HUE]. |
| *pnChroma:* | The pointer to the location for chroma saturation value, range [0 .. VME_MAX_CHROMA]. |
| *pnLum:* | The pointer to the location for luminance value, range [0 .. VME_MAX_LUMA]. |

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeGetBorderColorYUV

**#include "tekvme.h"**

**BOOL VmeGetBorderColorYUV(VmeHandle** *hMix,*
**VmeStage** *stage,*
**UINT*** *pnY,*
**UINT*** *pnU,*
**UINT*** *pnV*)

This function gets the color of the transition border in the specified mixer stage in YUV format.

**Parameters:**

*hMix:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeSecondStage
- VmeAllStages

*pnY:* The pointer to the location for the luminance component of the border color. The valid value range [256 .. 3670].

*pnU:* The pointer location for the first component of border color chrominance. The valid value range [256 .. 3840].

*pnV:* The pointer location for the second component of border color chrominance. The valid value range [256 .. 3840].

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeGetBorderWidth

**#include "tekvme.h"**

**INT VmeGetBorderWidth**(**VmeHandle** *vmeHandle*,
**VmeStage** *stage*)

This function gets the border width for the transition in the specified mixer stage.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**.

**Return Value:** This function returns the border width in range [0 .. VME_MAX_BORDER_WIDTH], or `-1` indicating an error. (For extended error information, use **GetLastError**.)

## VmeGetEdgeSoftness

**#include "tekvme.h"**

**INT VmeGetEdgeSoftness**(**VmeHandle** *vmeHandle*,
**VmeStage** *Stage*)

This function gets the edge softness for the specified mixer stage.

**Parameters:**

| | |
|---|---|
| *vmeHandle:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**. |

**Return Value:** This function returns the integer specifying edge softness in range [0 .. VME_MAX_SOFTNESS], or `-1` indicating an error. (For extended error information, use **GetLastError**.)

## VmeGetFadeLength

**#include "tekvme.h"**

**INT VmeGetFadeLength(VmeHandle** *vmeHandle*,
**VmeStage** *stage*)

This function gets the length in fields of the automatic fade initiated by *VmeStartFade* for the specified stage of the specified mixer.

**Parameters:**

| | |
|---|---|
| *vmeHandle:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**. |

**Return Value:** This function returns the length of the fade in number of video fields, or `-1` indicating an error. (For extended error information, use **GetLastError**.)

## VmeGetFadeMatteColor

**#include "tekvme.h"**

**BOOL VmeGetFadeMatteColor**(**VmeHandle** *vmeHandle,*
**VmeStage** *stage,*
**INT*** *pnHue,*
**INT*** *pnChroma,*
**INT*** *pnLum*)

This function gets the color of the “fade to matte” hardware at the end of the specified mixer stage.

**Parameters:**

| | |
|---|---|
| *vmeHandle:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**. |
| *pnHue:* | The pointer to the location for hue value, range [0 .. VME_MAX_HUE]. |
| *pnChroma:* | The pointer to the location for chroma saturation value, range [0 .. VME_MAX_CHROMA]. |
| *pnLum:* | The pointer to the location for luminance value, range [0 .. VME_MAX_LUMA]. |

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeGetFadeProgress

**#include "tekvme.h"**

**INT VmeGetFadeProgress**(**ResourceHandle** *resourceHandle*,
**VmeStage** *stage*)

This function gets the fader position in the specified stage of the specified mixer.

**Parameters:**

*resourceHandle:* The resource handle for a mixer returned by *VdrAllocateResource*.

*stage:* An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**.

**Return Value:** An integer in the range [0 .. VME_MAX_PROGRESS], or `-1` to indicate an error. (For extended error information, use **GetLastError**.)

**Comments:** Progress is in the *VmeFadeDown* direction with 0 being unfaded, and VME_MAX_PROGRESS being fully faded to matte.

## VmeGetHModulation

**#include "tekvme.h"**

**BOOL VmeGetHModulation(VmeHandle** *vmeHandle,*
**VmeStage** *stage,*
**VmeModulationType*** *pmodType,*
**INT*** *pnAmp,*
**FLOAT*** *peFreq,*
**INT*** *pnPhase,*
**INT*** *pnDeltaPhase*)

This function gets the horizontal edge modulation parameters for the specified mixer stage.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**.

*pmodType:* The pointer to the location for the modulation waveform type. Will receive one of the following:

- VmeNoWave
- VmeSineWave
- VmeTriangleWave
- VmeSawtoothWave
- VmeSquareWave

*pnAmp:* The pointer to the location for the peak-to-peak amplitude of the waveform. Units are not calibrated and are arbitrarily chosen to be in the range [0 .. VME_MAX_MOD_AMP].

*peFreq:* The pointer to the location for the frequency for the waveform. A value of `1.0` corresponds to one cycle per active line width.

*pnPhase:* The pointer to the location for the starting phase of the waveform. In rotation units, range
[0 .. VME_MAX_ROTATION].

*pnDeltaPhase:* Pointer to the location for the frame-to-frame phase change. In rotation units, range [0 .. VME_MAX_ROTATION].

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeGetKey

**#include "tekvme.h"**

**BOOL VmeGetKey**(**VmeHandle** *vmeHandle,*
**VmeStage** *stage,*
**INT*** *pnClipLevel,*
**FLOAT*** *peGain,*
**VmeKeyInvertMode*** *pInvertMode,*
**VmeKeyEdgeMode*** *pEdgeMode*)

This function gets the clip and gain values for the video luminance key on the specified mixer stage.

**Parameters:**

| | |
|---|---|
| *vmeHandle:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**. |
| *pnClipLevel:* | The pointer to the location for the clip level in range from [VME_MIN_CLIP_LEVEL .. VME_MAX_CLIP_LEVEL]. |
| *peGain:* | The pointer to the location for the gain value for the key in range from [VME_MIN_GAIN_VALUE .. VME_MAX_GAIN_VALUE]. |
| *pInvertMode:* | The pointer to the location for the inverted key mode value. Will receive either **VmeNormalKey** or **VmeInvertedKey**. |
| *pEdgeMode:* | The pointer to the location for the key edge value. Will receive either **VmeMultiplicativeKey** or **VmeAdditiveKey**. |

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeGetMajorVersion

**#include "tekvme.h"**

**UINT VmeGetMajorVersion(ConnectHandle** *connHandle*)

This function returns the major version number of the Video Mix Effects library. The major number is the portion before the decimal point. For example, if the library version number is 1.23, the call would return **1**.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the integer major version number of the library.

## VmeGetMinorVersion

**#include "tekvme.h"**

**UINT VmeGetMinorVersion**(**ConnectHandle** *connHandle*)

This function returns the minor version number of the Video Mix Effects library. The minor number is the portion after the decimal point. For example, if the library version number is 1.23, the call would return **23**.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

**Return Value:** This function returns the integer minor version number of the library.

## VmeGetMultipliers

**#include "tekvme.h"**

**BOOL VmeGetMultipliers(VmeHandle** *vmeHandle,*
**VmeStage** *stage,*
**INT*** *pnHMul,*
**INT*** *pnVMul*)

This function gets the horizontal and vertical multipliers for the specified stage of the specified mixer.

**Parameters:**

| | |
|---|---|
| *vmeHandle:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**. |
| *pnHMul:* | The pointer to the location for the horizontal integer multiplier value, range [0 .. VME_MAX_MULTIPLIER]. |
| *pnVMul:* | The pointer to the location for the vertical integer multiplier value, range [0 .. VME_MAX_MULTIPLIER]. |

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** These multipliers can turn a simple horizontal wipe into “venetian blinds” or a corner wipe into a checkerboard, for example.

**See Also:** *VmeSetMultipliers* on page 548.

## VmeGetPosition

**#include "tekvme.h"**

**BOOL VmeGetPosition(VmeHandle** *vmeHandle*,
**VmeStage** *stage*,
**INT*** *pnX*,
**INT*** *pnY*)

This function gets the position of the transition in the specified mixer stage.

**Parameters:**

| | |
|---|---|
| *vmeHandle:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**. |
| *pnX:* | The pointer to the location for position of the start of the transition. |
| *pnY:* | The pointer to the location for position of the start of the transition. |

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**See Also:** *VmeSetPosition* on page 549.

## VmeGetRotation

**#include "tekvme.h"**

**INT VmeGetRotation**(**VmeHandle** *vmeHandle*,
**VmeStage** *stage*)

This function gets the rotation of the transition in the specified mixer stage.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**.

**Return Value:** This function returns the rotation value in the range [0 .. VME_MAX_ROTATION].

## VmeGetTransitionLength

**#include "tekvme.h"**

**INT VmeGetTransitionLength(VmeHandle** *vmeHandle*,
**VmeStage** *stage*)

This function gets the length in fields of the automatic transition initiated by *VmeStartTransition* for the specified stage of the specified mixer.

**Parameters:**

| | |
|---|---|
| *vmeHandle:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**. |

**Return Value:** This function returns the length of the transition in number of video fields.

## VmeGetTransitionProgress

**#include "tekvme.h"**

**INT VmeGetTransitionProgress**(**ResourceHandle** *resourceHandle*,
**VmeStage** *stage*)

This function gets the transition position in the specified stage of the specified mixer.

**Parameters:**

*resourceHandle:* The resource handle for a mixer returned by *VdrAllocateResource*.

*stage:* An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**.

**Return Value:** An integer in the range [0 .. VME_MAX_PROGRESS], or `-1` to indicate an error. (For extended error information, use **GetLastError**.)

**Comments:** Progress is in the VmeBgToFg direction, with 0 being fully background, and VME_MAX_PROGRESS being fully foreground.

## VmeGetTransitionType

**#include "tekvme.h"**

**TransitionType VmeGetTransitionType**(**VmeHandle** *vmeHandle*,
**VmeStage** *stage*)

This function gets the transition type from the given stage of the mix context.

**Parameters:**

| | |
|---|---|
| *vmeHandle:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**. |

**Return Value:** This function returns the **TransitionType** set in the context, or `NoTransition` if none is set.

## VmeGetVModulation

**#include "tekvme.h"**

**BOOL VmeGetVModulation(VmeHandle** *vmeHandle,*
**VmeStage** *stage,*
**VmeModulationType*** *pModType,*
**INT*** *pnAmp,*
**FLOAT*** *peFreq,*
**INT*** *pnPhase,*
**INT*** *pnDeltaPhase*)

This function gets the border modulation parameters for the specified mixer stage.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**.

*pModType:* The pointer to the location for the modulation waveform type. Will receive one of the following:

- VmeNoWave
- VmeSawtoothWave
- VmeSineWave
- VmeSquareWave
- VmeTriangleWave

*pnAmp:* The pointer to the location for the peak-to-peak amplitude of the waveform. Units are not calibrated and are arbitrarily chosen to be in the range [0 .. VME_MAX_MOD_AMP].

*peFreq:* The pointer to the location for the floating point frequency value for the waveform in the range [0.0 .. VME_MAX_MOD_FREQ]. 1.0 corresponds to one cycle per active field height.

*pnPhase:* The pointer to the location for the waveform's starting phase. In rotation units, range [0 .. VME_MAX_ROTATION].

*pnDeltaPhase:* Pointer to the location for the frame-to-frame phase change. In rotation units, range [0 .. VME_MAX_ROTATION].

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeResetFade

**#include "tekvme.h"**

**BOOL VmeResetFade(VmeHandle** *vmeHandle*,
**VmeStage** *stage*)

This function sets the fader on the output of the given mixer stage back to its default values, which specify a simple fade to black with length of 1 second (50 or 60 fields).

**Parameters:**

| | |
|---|---|
| *vmeHandle:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**. |

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeResetTransition

**#include "tekvme.h"**

**BOOL VmeResetTransition(VmeHandle** *vmeHandle*,
**VmeStage** *stage*)

This function sets the transition of the given mixer stage back to its default values. This will be a simple full range dissolve in the **VmeBgToFg** direction with duration of 1 second (50 or 60 fields).

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**.

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeSetAspect

**#include "tekvme.h"**

**BOOL VmeSetAspect**(**VmeHandle** *hMix*,
**VmeStage** *stage*,
**INT** *nAspect*)

This function sets the aspect ratio number for the transition set in the mix context specified by the given handle. The aspect ratio number is a fixed point value, with `1000` (defined by `VME_NORMAL_ASPECT`) being 1:1 (horizontal:vertical). Twice that value will give a 2:1 aspect, while half will result in 1:2.

**Parameters:**

| | |
|---|---|
| *hMix:* | Handle to the mix context allocated by VmeAllocateContext. |
| *stage:* | Value specifying a mixer stage. Must be VmeAllStages or VmeFirstStage. |
| *nAspect:* | The aspect ratio fixed point number. |

**Return Value:** OK if the specified values are acceptable; otherwise, ERROR. To get extended error information, user the **GetLastError** function.

**Comments:** Only values greater than zero have any meaning, and some transitions may limit this value to certain ranges of aspect. For example, circles won't work outside the 2:1 or 1:2 range. This aspect value will have no effect on transitions that have no aspect (like horizontal wipes).

**See Also:** *VmeGetAspect* on page 517,
*VmeGetRotation* on page 531, and
*VmeSetRotation* on page 550.

## VmeSetBorderColor

**#include "tekvme.h"**

**BOOL VmeSetBorderColor(VmeHandle** *vmeHandle*,
**VmeStage** *stage*,
**INT** *nHue*,
**INT** *nChroma*,
**INT** *nLum*)

This function sets the color of the wipe transition border in the specified mixer stage.

**Parameters:**

| | |
|---|---|
| *vmeHandle:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be **VmeFirstStage**. |
| *nHue:* | The hue value, range [0 .. VME_MAX_HUE]. |
| *nChroma:* | The chroma saturation value, range [0 .. VME_MAX_CHROMA]. |
| *nLum:* | The luminance value, range [0 .. VME_MAX_LUMA]. |

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeSetBorderColorYUV

**#include "tekvme.h"**

**BOOL VmeSetBorderColorYUV**(**VmeHandle** *hMix*,
**VmeStage** *stage*,
**UINT** *nY*,
**UINT** *nU*,
**NINT** *nV*)

This function sets the color of the transition border in the specified mixer stage using the supplied YUV settings.

**Parameters:**

| | |
|---|---|
| *hMix:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be one of the following:<br>• VmeFirstStage<br>• VmeSecondStage<br>• VmeAllStages |
| *nY:* | The luminance component of the border color to be set. The valid value range [256 .. 3760]. |
| *nU:* | The first component of border color chrominance to be set. The valid value range [256 .. 3840]. |
| *nV:* | The second component of border color chrominance to be set. The valid value range [256 .. 3840]. |

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeSetBorderWidth

**#include "tekvme.h"**

**BOOL VmeSetBorderWidth(VmeHandle** *vmeHandle*,
**VmeStage** *stage*,
**INT** *nWidth*)

This function sets the border width for the transition in the specified mixer stage.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*nWidth:* The integer width for the border. Border width units are not calibrated and are arbitrarily chosen to be in the range [0 .. VME_MAX_BORDER_WIDTH].

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeSetEdgeSoftness

**#include "tekvme.h"**

**BOOL VmeSetEdgeSoftness(VmeHandle** *vmeHandle,*
**VmeStage** *stage,*
**INT** *nSoftness*)

This function sets the edge softness for the specified mixer stage.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*nSoftness:* The integer which specifies the edge softness. It must be in the range [0 .. VME_MAX_SOFTNESS].

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The softness is specified using an arbitrary scale, where 0 is maximum edge hardness and VME_MAX_SOFTNESS is the maximum edge softness.

## VmeSetFadeLength

**#include "tekvme.h"**

**BOOL VmeSetFadeLength**(**VmeHandle** *vmeHandle*,
**VmeStage** *stage*,
**INT** *nFields*)

This function sets the length in fields of the automatic fade initiated by *VmeStartFade* for the specified stage of the specified mixer.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*nFields:* The length of the fade in number of video fields.

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeSetFadeMatteColor

**#include "tekvme.h"**

**BOOL VmeSetFadeMatteColor**(**VmeHandle** *vmeHandle*,
**VmeStage** *stage*,
**INT** *nHue*,
**INT** *nChroma*,
**INT** *nLum*)

This function sets the color of the “fade to matte” hardware at the end of the specified mixer stage.

**Parameters:**

| | |
|---|---|
| *vmeHandle:* | A mix context handle. |
| *stage:* | An enumerated type specifying the stage of the mixer. This must be one of the following:<br>• VmeFirstStage<br>• VmeAllStages |
| *nHue:* | The hue value, range [0 .. VME_MAX_HUE]. |
| *nChroma:* | The chroma saturation value, range [0 .. VME_MAX_CHROMA]. |
| *nLum:* | The luminance value, range [0 .. VME_MAX_LUMA]. |

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** If no value is set, the fade is the default color, which is black.

## VmeSetFadeProgress

**#include "tekvme.h"**

**BOOL VmeSetFadeProgress**(**ResourceHandle** *resourceHandle*,
**VmeStage** *stage*,
**INT** *nProgress*)

This function sets the fader in the specified stages of the specified mixer to the specified position.

**Parameters:**

*resourceHandle:* The resource handle for a mixer returned by *VdrAllocateResource*.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*nProgress:* An integer in the range [0 .. VME_MAX_PROGRESS], where 0 is the unfaded or fade up state and VME_MAX_PROGRESS is the faded or fade down state.

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The mixer must have a mix context applied to get anything other than the default. Default action is fade to black. Progress is in the **VmeFadeDown** direction, with 0 being no fade, or faded up and VME_MAX_PROGRESS being fully faded or faded down.

## VmeSetHModulation

**#include "tekvme.h"**

**BOOL VmeSetHModulation**(**VmeHandle** *vmeHandle,*
**VmeStage** *stage,*
**VmeModulationType** *modType,*
**INT** *nAmp,*
**FLOAT** *eFreq,* INT *nPhase,*
**INT** *nDeltaPhase*)

This function sets the horizontal edge modulation parameters for the specified mixer stage.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*modType:* The modulation waveform type. One of the following:

- VmeNoWave
- VmeSineWave
- VmeTriangleWave
- VmeSawtoothWave
- VmeSquareWave

*nAmp:* The peak-to-peak amplitude of the waveform. Units are not calibrated and are arbitrarily chosen to be in the range [0 .. VME_MAX_MOD_AMP].

*eFreq:* A floating point frequency value for the waveform in the range [0.0 .. VME_MAX_MOD_FREQ]. A value of `1.0` corresponds to one cycle per active line width.

*nPhase:* The starting phase of the waveform. Specified in rotation units, mapped to range [0 .. VME_MAX_ROTATION].

*nDeltaPhase:* The frame-to-frame change in phase. Specified in rotation units, mapped to range [0 .. VME_MAX_ROTATION].

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The optional modulation waveform can be sine, sawtooth, or square wave with specified amplitude, frequency, and phase. A change in phase from one frame to the next can also be specified to create a “running” waveform.

## VmeSetKey

**#include "tekvme.h"**

**BOOL VmeSetKey**(**VmeHandle** *vmeHandle,*
**VmeStage** *stage,*
**INT** *nClipLevel,*
**FLOAT** *eGain,*
**VmeKeyInvertMode** *invertMode,*
**VmeKeyEdgeMode** *edgeMode*)

This function sets the clip and gain values for the video luminance key on the specified mixer stage.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*nClipLevel:* A clip level in range [0 .. VME_MAX_CLIP_LEVEL].

*eGain:* A gain value for the key in range [VME_MIN_GAIN_VALUE .. VME_MAX_GAIN_VALUE].

*invertMode:* Enumerated type specifying the key invert mode. This must be one of the following:

- VmeNormalKey
- VmeInvertedKey

*edgeMode:* Enumerated type specifying the key edge mode. This must be one of the following:

- VmeAdditiveKey
- VmeMultiplicativeKey

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeSetMultipliers

**#include "tekvme.h"**

**BOOL VmeSetMultipliers**(**VmeHandle** *vmeHandle,*
**VmeStage** *stage,*
**INT** *nHMul,*
**INT** *nVMul*)

This function sets the horizontal and vertical multipliers for the specified stage of the specified mixer.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*nHMul:* The horizontal integer multiplier value. Limited to the range [1 .. VME_MAX_MULTIPLIER].

*nVMul:* The vertical integer multiplier value. Limited to the range [1 .. VME_MAX_MULTIPLIER].

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** These multipliers can turn a simple horizontal wipe into “venetian blinds” or a corner wipe into a checkerboard, for example.

## VmeSetPosition

**#include "tekvme.h"**

**BOOL VmeSetPosition**(**MixHandle** *vmeHandle,*
**VmeStage** *stage,*
**INT** *nX,*
**INT** *nY*)

This function sets the position of the transition in the specified mixer stage.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*nX:* The horizontal position of the start of the transition, range:

[-2 * VME_MAX_SCREEN_HEIGHT..
2* VME_MAX_SCREEN_HEIGHT].

*nY:* The vertical position of the start of the transition, range:

[-2* VME_MAX_SCREEN_HEIGHT..
2* VME_MAX_SCREEN_HEIGHT].

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** This function allows for the positioning of a box wipe, for example. Positions are defined in arbitrary, square units with (0, 0) in the center of the screen and with VME_MAX_SCREEN_HEIGHT representing the full height of the video screen. Thus ((2*VME_MAX_SCREEN_HEIGHT) / 3, VME_MAX_SCREEN_HEIGHT / 2) is the upper right corner. (-(2*VME_MAX_SCREEN_HEIGHT) / 3, -VME_MAX_SCREEN_HEIGHT / 2) is the lower left corner.

## VmeSetRotation

**#include "tekvme.h"**

**BOOL VmeSetRotation(VmeHandle** *vmeHandle*,
**VmeStage** *stage*,
**INT** *nRotation*)

This function sets the rotation of the transition in the specified mixer stage.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*nRotation:* The integer number mapped to the range [0 .. VME_MAX_ROTATION].

**Return Value:** TRUE if successful; otherwise, FALSE.

**Comments:** Rotations are specified in arbitrary units with positive increasing counter-clockwise, and with a full 360-degree circle being represented as `VME_MAX_ROTATION`.

## VmeSetTransitionLength

**#include "tekvme.h"**

**BOOL VmeSetTransitionLength**(**VmeHandle** *vmeHandle*,
**VmeStage** *stage*,
**INT** *nFields*)

This function sets the length in fields of the automatic transition initiated by *VmeStartTransition* for the specified stage of the specified mixer.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*nFields:* The length of the transition in number of video fields.

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## VmeSetTransitionProgress

**#include "tekvme.h"**

**BOOL VmeSetTransitionProgress**(**ResourceHandle** *resourceHandle*,
**VmeStage** *stage*,
**INT** *nProgress*)

This function sets the transition in the specified stages of the specified mixer to the specified position. The mixer must have a mix context applied to get anything other than the default dissolve.

**Parameters:**

*resourceHandle:* The resource handle for a mixer returned by *VdrAllocateResource*.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*nProgress:* An integer in the range [0 .. VME_MAX_PROGRESS].

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** Progress is in the VmeBgToFg direction, with `0` being fully background and `VME_MAX_PROGRESS` being fully foreground.

## VmeSetTransitionType

**#include "tekvme.h"**

**BOOL VmeSetTransitionType**(**VmeHandle** *vmeHandle*,
**VmeStage** *stage*,
**TransitionType** *type*)

This function sets the specified transition in the mix context.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*type:* One of the defined **TransitionTypes** allowed for the mixer type of the context.

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The transition will have default parameters if the context is new, otherwise it will try and use the other settings (such as length, border color, modulation, etc.) already in the context.

Please see the **TransitionType** definition in the header file *tekvme.h* for a complete list of the supported transitions.

## VmeSetVModulation

**#include "tekvme.h"**

**BOOL VmeSetVModulation**(**VmeHandle** *vmeHandle,*
**VmeStage** *stage,*
**VmeModulationType** *modType,*
**INT** *nAmp,*
**FLOAT** *eFreq,*
**INT** *nPhase,*
**INT** *nDeltaPhase*)

This function sets the vertical edge modulation parameters for the specified mixer stage.

**Parameters:**

*vmeHandle:* A mix context handle.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*modType:* The modulation waveform type. One of the following:

- VmeNoWave
- VmeSineWave
- VmeTriangleWave
- VmeSawtoothWave
- VmeSquareWave

*nAmp:* The peak-to-peak amplitude of the waveform. Units are not calibrated, and are arbitrarily chosen to be in the range [0 .. VME_MAX_MOD_AMP].

*eFreq:* A frequency value for the waveform. A value of `1.0` corresponds to one cycle per field.

*nPhase:* The starting phase of the waveform. Specified in rotation units, mapped to range [0 .. VME_MAX_ROTATION].

*nDeltaPhase:* The frame-to-frame phase change. Specified in rotation units, mapped to range [0 .. VME_MAX_ROTATION].

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The optional modulation waveform can be sine, sawtooth or square wave with specified amplitude, frequency, and phase. A change in phase from one frame to the next can also be specified to create a “running” waveform.

## VmeStartFade

**#include "tekvme.h"**

**BOOL VmeStartFade(ResourceHandle** *resourceHandle,*
**VmeStage** *stage,*
**VmeFadeDirection** *dir*)

This function starts an automatic fade in the specified stages of the specified mixer.

**Parameters:**

*resourceHandle:* The resource handle for a mixer returned by *VdrAllocateResource*.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*dir:* An enumerated type specifying the direction of the fade. This must be one of the following:

- VmeFadeUp
- VmeFadeDown

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** Each mixer stage has a fader on its output which can be run independent of whatever transition is on its input.

The mixer must have had a mix context applied to get anything other than the default. Default action is a one-second fade-to-black (50 or 60 fields).

## VmeStartFadeAtGenTc

**#include "tekvme.h"**

**Comments:** New command; engineering specification not yet available.

## VmeStartTransition

**#include "tekvme.h"**

**BOOL VmeStartTransition**(**ResourceHandle** *resourceHandle*,
**VmeStage** *stage*,
**VmeTransitionDirection** *dir*)

This function starts an automatic transition in the specified stages of the specified mixer.

**Parameters:**

*resourceHandle:* The resource handle for a mixer returned by *VdrAllocateResource*.

*stage:* An enumerated type specifying the stage of the mixer. This must be one of the following:

- VmeFirstStage
- VmeAllStages

*dir:* An enumerated type specifying the direction of the transition. This must be one of the following:

- VmeBfToFg
- VmeFgToBg

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** The mixer must have a mix context applied to get anything other than the default. Default action is a one-second dissolve (50 or 60 fields).

## VmeStartTransitionAtGenTc

**#include "tekvme.h"**

**Comments:** New command; engineering specification not yet available.

# The Xfr library

The TekXfr library allows the transfer of a movie at the same time that it is being recorded, or the playout of a movie as it is being transferred. This is commonly called streaming.

## XfrAbort

**#include "tekxfr.h"**

**BOOL XfrAbort(XfrToken** *t***)**

This function requests that a transfer in progress be stopped, or releases the XfrToken *t* in a streaming transfer that is already done.

**Parameters:**

*t:* The transfer token returned by a call to *XfrRequest*.

**Return Value:** TRUE if successful; otherwise, FALSE. (If the Abort is issued on a command that has already been aborted, it will return a FALSE status.) (For extended error information, use **GetLastError**.)

**Comments:** An error is returned only in the case where a transfer is unknown to the system on which the request is made. The execution is a “best effort”, meaning that it stops the transfer only if it can.

## XfrGetActiveTokens

**#include "tekxfr.h"**

**BOOL XfrGetActiveTokens**(**ConnectHandle** *connHandle,*
**UINT** *maxEntries,*
**XfrToken** *idList[],*
**UINT*** *returnCount*)

This function returns an array of the transfers currently occurring on the system it is requested on. This allows applications to provide in-transit status. With this list, the application can then call *XfrGetStatus* for all of the transfers currently occurring or in the status cache.

**Parameters:**

| | |
|---|---|
| *connHandle:* | The value returned from *RemOpenConnection* or LOCAL_CONNECTION. |
| *maxEntries:* | The maximum number of XfrTokens that will fit into the idList array. |
| *idList[]:* | Pointer to an array that will receive the XfrTokens which are currently active. |
| *returnCount:* | Pointer to a variable which will receive the number of tokens that were written to the idList array. This value will always be less than or equal to *maxEntries*. |

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

**Comments:** If *returnCount* is specified, the number in the list will be returned.

## XfrGetStatus

**#include "tekxfr.h"**

**BOOL XfrGetStatus(XfrToken** *t*,
**XfrStatus*** *status***)**

This function returns the current status of a transfer in progress, or a transfer that was completed earlier.

**Parameters:**

*t:* The transfer token returned by this call to *XfrRequest*.

*status:* A pointer to the variable to receive the status information.

**Return Value:** TRUE if successful; otherwise, FALSE. (For extended error information, use **GetLastError**.)

## XfrRequest

**#include "tekxfr.h"**

**XfrToken XfrRequest**(**ConnectHandle** *connHandle,*
**CHAR*** *sourceUml,*
**UINT** *destCount,*
**CHAR**** *destUmlList*)

This function initiates a transfer between a single data source described by the *sourceUml*, and a list of one or more destinations described by the *destUmlList*.

**Parameters:**

*connHandle:* The value returned from *RemOpenConnection* or LOCAL_CONNECTION.

*destCount:* Must be `1`.

*destUmlList:* UML entry specifying the destination for the transfer. (See *Comments* below.)

**Return Value:** This function returns a token identifying the transfer that can be used in subsequent operations. It returns a NULL if not successful. (For extended error information, use **GetLastError**.)

**Comments:** This call is asynchronous to the copy itself and returns immediately after initiation. All further operations are carried out using the **XfrToken**. The current software release supports only one destination per host.

UMLs (Uniform Media Locators) are similar in concept to URLs (Universal Resource Locators) used with HTTP on the Internet. They are meant to provide a complete description of the source or destination of a transfer over Fibre Channel.

The generic form for any UML is specified as:

`<host>/<type>/<typeSpecificInformation>[?<options>]`

The `<host>` portion consists of the Windows NT computer name of the source or destination computer. The host must be available on the Fibre Channel network with the other associated machines; if the host is not specified, it will default to LOCALHOST.

The `<type>` portion represents the type of transfer to be performed at the source or destination. Presently, *ExplodedFile* is the only type defined for the external API.

The `<typeSpecificInformation>` portion is described by the dataset, the group, and the name of the clip.

The bracketed `?<options>` piece indicates a group of optional arguments that will allow you to specify which fields will be copied from the clip and also to designate streaming priority.

**See Also:** The “Transferring Media with Fibre Channel” chapter of the *Profile SDK User Manual* contains more information on UMLs.

Chapter 3

# Profile Serial Communication Protocol

The Profile protocol provides a method for controlling a Profile system remotely from another machine (via an RS-422 connection, for example.) This appendix describes the Profile command protocol and provides an example of a typical insert/edit session that uses these commands. The commands are divided into the following categories:

- *System access commands (0x00)*
- *PLS serial protocol commands (0x0E)*
- *Port access commands (0x10)*
- *PDR movie access commands (0x21)*
- *Attached movie access commands (0x30)*
- *Stored media access commands (0x40)*
- *Attached media access commands (0x50)*
- *Resource access commands (0x60)*
- *Event access commands (0x70)*
- *Video mix effects access commands (0x80)*
- *Streaming transfer access commands (0x90)*

***NOTE: Stored movie access commands (0x20) have been obsoleted from the Profile protocol and do not appear in this manual.***

See the *Software Development Kit User Manual* for an explanation of the Profile conceptual model, including ports, resources, stored movies and attached movies, and stored media files and attached media files.

The interface for accessing and working with the Profile protocol is an application called ProLink, which can be invoked by double clicking on the ProLink icon.

In the ProLink application:

- From the **File** menu, you can exit the program by selecting **Exit**.
- The logger writes in TXT format to the file *c:\profile\logs\prolinkx.yyy*, where *x* is the port number used by the Prolink session and *yyy* is either `000` or `001`.

# Command structure

Commands issued by the controller have this form (each cell is one byte):

| Stx | Byte count | Cmd1 | Cmd2 | Data1 | Data2 | ... | DataN | Checksum |
|---|---|---|---|---|---|---|---|---|

Command fields are defined below:

| | | |
|---|---|---|
| Stx | Start of text, always 0x02. | |
| Bytecount | The number of bytes between Bytecount and Checksum (exclusive), ranges from 2- 255. | |
| Cmd1 | The major command type, one of: | |
| | 0x00 | System Access |
| | 0x10 | Port Access |
| | 0x21 | PDR Movie Access |
| | 0x30 | Attached Movie Access |
| | 0x40 | Stored Media Access |
| | 0x50 | Attached Media Access |
| | 0x60 | Resource Access |
| | 0x70 | Event Access |
| | 0x80 | Mix Effects Access |
| Cmd2 | The individual command number, unique to the major command type. | |
| Data1 .. DataN | Data needed to complete the command. | |
| Checksum | The least significant byte of the two's complement of the sum of all bytes from Bytecount to Checksum, but excluding Bytecount and Checksum. | |

A successful command cycle between a controller and a Profile includes these events:

- the controller issues the command to the Profile;
- the Profile receives the command and verifies that the command has the structure described above (including leading Stx value and correct checksum); and
- the Profile replies to the controller.

# Reply structure

If a command is incorrectly structured, the Profile replies with a *Nak*:

| | | |
|---|---|---|
| byte0 | 0x05 | |
| byte1 | Bitmap, including one or more of the following, indicating the problem in the command: | |
| | 0x04 | Unspecified error (checksum, invalid parameters, unable to execute, etc.) |
| | 0x10 | Parity error |
| | 0x20 | Over-run |
| | 0x40 | Framing error (no STX header) |
| | 0x80 | Time-out |

When a command is correctly structured and requests status information that is immediately available (see commands for details), the Profile replies with:

| | |
|---|---|
| byte0 | 0x02 |
| byte1 | Byte count |
| byte2 | Cmd1 value (from the command) |
| byte3 | Cmd2 value (from the command) |
| byte4..N | (data1...) reply values (see commands for details) |
| byte N+1 | Checksum (as described above) |

If any command fails, the “generic” packet description will have an additional four data bytes following the “error return code” (usually nothing but a 1). These four bytes are the **GetLastError** code return value.

When a command is correctly structured but takes longer than 10ms to complete, the Profile replies immediately with an *Ack*:

| | | |
|---|---|---|
| byte0 | 0x04 | |
| byte1 | Bitmap, either (00) or including one of the following error conditions: | |
| | bit0 | Command not implemented |
| | bit1 | Port busy |
| | bit2 | Incorrect number of bytes |
| | bit3 | Invalid transaction number |
| | bit4 | Invalid parameter |
| | bit5-6 | Reserved |
| | bit7 | RS422/RS232 event notification |
| byte2 | Transaction number assigned to this command | |

For a correctly structured command, the Profile continues transaction processing after sending the *Ack*. The reply can be retrieved using the **00.00 Get Transaction Reply** command.

When **00.00 Get Transaction Reply** is received for a completed valid transaction number, the Profile transaction reply is:

| | |
|---|---|
| byte0 | 0x02 |
| byte1 | Byte count |
| byte2 | Cmd1 value (from the command which started the transaction) |
| byte3 | Cmd2 value (from the command which started the transaction) |
| byte4..N | Reply values (see commands for details) |
| byte N+1 | Checksum (as described above) |

When **00.00 Get Transaction Reply** is received for an incomplete valid transaction number, the Profile replies with an additional *Ack* as outlined above.

# Data representation

In the command stream, floating-point numbers are represented in IEEE single-precision floating-point format (32 bits). The least significant bits are represented in the first byte transmitted, while the most significant bits are represented in the fourth byte transmitted.

For 32-bit signed and unsigned integers, handles, and bitmasks, the least significant bits are represented in the first byte transmitted, while the most significant bits are represented in the fourth byte transmitted.

Timecodes are represented as:

| | |
|---|---|
| byte0 | Fields |
| byte1 | Seconds |
| byte2 | Minutes |
| byte3 | Hours (bit 6 indicating drop-frame when set) |

The modification date and time structure (unsigned integer, 4 bytes) is as follows:

| | |
|---|---|
| byte0-4 | Day |
| byte5-8 | Month |
| byte9-15 | Year offset from 1980 |
| byte16-20 | Half seconds |
| byte21-26 | Minutes |
| byte27-31 | Hours |

Strings used for names are always null-terminated ASCII characters. The null is included in the transmission packets to simplify parsing.

The common movie format provides a structure to the media files that comprise the movies so that a single media file can participate in many movies.

# A typical serial session

The following example shows the commands issued during a typical insert-edit session (for Ack replies, the entire reply is shown; but for other replies, Stx, Bytecount, and Checksum are omitted):

**Table 24. Typical serial session interaction**

| Command | Command name | Action | Reply |
|---|---|---|---|
| 10.01. s.a.m.p.l.e. | Open Port | Open a port with sample.cfg configuration file [a] | 04.00.t0 (where t0 is the transaction number assigned by the disk recorder) |
| 00.00.t0 | | Retrieve handle for port (waiting) ... | 04.00.t0 |
| 00.00.t0 | | Retrieve handle for port (done waiting) ... | 10.01.t0.pp.pp.pp.pp (where pp.pp.pp.pp is the handle for the port) |
| 30.02.23.31.00.00.00.00.0 0.bc.a5.01.00.pp.pp.pp.pp. | Attach Movie with Marks | Attach movie "#1" (23.31.00), setting mark-in field to 00:00:00:00 (00.00.00.00) and mark-out field to 00:30:00:00 (bc.a5.01.00) to the port pp.pp.pp.pp. | 04.00.t1 (where t1 is the transaction number assigned by the disk recorder) |
| 00.00.t1 | | Retrieve handle for movie (waiting) ... | 04.00.t1 |
| 00.00.t1 | | Retrieve handle for movie (done waiting) | 30.02.t1.hh.hh.hh.hh (where hh.hh.hh.hh is the handle for the movie) |
| 30.00.hh.hh.hh.hh.01.00 | Get Attached Movie Status | Ask for status for movie starting position | 04.00.t2 (where t2 is the transaction number assigned) |
| 00.00.t2 | | Retrieve status for movie (waiting) ... | 04.00.t2 |
| 00.00.t2 | | Retrieve status for movie (done waiting) | 30.00.t2.00.01.00.07.00.00 (for example, start field 0x700) |
| 10.0a.pp.pp.pp.pp.44.54.0 0.00 (0x5444= 0x700+0x4d44) | Set Position | Set position to field 5 min 30 sec. into movie (0x4d44 fields from start) | 04.00.t3 (where t3 is the transaction number assigned) |
| 00.00.t3 | | Check for position set (waiting)... | 04.00.t3 |
| 00.00.t3 | | Check for position set (done waiting) | 10.0a.t3.00 |
| 10.07.pp.pp.pp.pp | Cue Record | Cue-record | 04.00.t4 (where t4 is the transaction number assigned) |
| 10.00.pp.pp.pp.pp.01.00 | Get Port Status | Wait for the state to change to cue-record | 10.00.01.00.01 (cue-play) |
| 10.00.pp.pp.pp.pp01.00 | Get Port Status | Done waiting (skip the transaction reply for t4) | 10.00.01.00.02 (cue-record) |

**Table 24. Typical serial session interaction (Continued)**

| Command | Command name | Action | Reply |
|---|---|---|---|
| 10.08.pp.pp.pp.pp | Shuttle Record | Preroll the source, and one frame before playing the first frame of source, record | 04.00.t5 (where t5 is the transaction number assigned) |
| 10.04.pp.pp.pp.pp | Cue Play | After the last frame of source has started playing, cue-play (ends recording) | 04.00.t6 (where t6 is the transaction number assigned) |
| 30.03.hh.hh.hh.hh | Detach Movie | Detach movie | 04.00.t7 (where t7 is the transaction number assigned) |
| 10.02 | Close Port | Close the port (skip the transaction replies for t5, t6, and t7) | 04.00.t8 |

a. This allocates all desired resources and sets up default and scheduled events.

***NOTE: In the command listing which follows, each serial command name cross-references the equivalent API command name.***

# Configuration files for Profile protocol

Profile protocol implements Named Configuration Space (NCS) as a mechanism to ease Profile configuration. Using NCS, an application can send one command, Open Port (10.01), with a parameter specifying a configuration to use, and the Profile video server will handle all the resource allocation and crosspoints specified by that named configuration.

In versions 4.0 and later, the named configuration corredsponds to a channel name that is defined in Configuration Manager. Once a channel is defined, it can be used by any application, including Pronet, to assign resources to that application. The Resource Monitor (resmon) reports resource conflicts as they occur.

In versions of Profile System Software prior to version 4.0, the named configuration corresponds to a configuration file that contains all the required configuration information. This frees the application from the task of allocating resources manually, but it does require proper understanding of the configuration files.

The following section explains the format of the configuration files used with versions of Profile System Software prior to 4.0.

## Overview of configuration files

Configuration files are simple text files, containing three types of lines. The lines are parsed on a column basis as follows:

- if column 1 contains anything besides a number from 0-9, the line is ignored as a **comment** line;
- if column 7 contains a ‘C’ or ‘c’, the line is treated as a **crosspoint** line; and
- all other lines are treated as **resource** lines.

Both resource allocation lines and crosspoint lines share a similar format:

```
xx-yy[,][func1][func2]
```

***NOTE: Although resource and crosspoint lines look similar, they have very different meanings. This is a frequent source of confusion and error. Make sure whether you are editing a resource line or a crosspoint line!***

## Resource lines

Resource lines allocate a resource for use by a port. The following is the column-by-column layout of resource lines:

| | | |
|---|---|---|
| column 1-2: | resource type, which must be one of the following: | |
| | 00 | audio codec |
| | 01 | JPEG codec |
| | 02 | video input |
| | 03 | video output |
| | 04 | LTC input |
| | 05 | VITC input |
| | 06 | timecode generator |
| | 07 | LTC output |
| | 08 | VITC output |
| | 09 | timecode recorder |
| | 10 | audio output |
| | 11 | video combiner (mix effect) |
| | 12 | MPEG encoder |
| | 13 | MPEG decoder |
| | 14 | audio input |
| | 15 | DVCPRO25 codec |
| | 16 | DVCPRO50 codec |
| column 3: | separator (-) | |
| column 4-5: | resource number | |

This specifies which resource of the specified type is desired. Resources of a given type are numbered starting at 0. Thus, to select the third timecode recorder, specify ‘02’ in columns 4-5.

For inputs and outputs spread across multiple cards, devices are scanned along the Profile EISA bus starting at slot 0. Viewed from the back of the Profile, numbers increment from right to left. Thus, in a Profile with two serial digital I/O cards, video input 00 represents the upper BNC connector on the rightmost serial digital card, while video input 03 represents the second BNC connector on the leftmost serial digital card.

In Profile systems with a mix of video I/O cards, the video cards are scanned in the following order:

- serial digital component
- serial digital component with embedded audio
- component analog
- composite analog
- serial digital composite

For example, in a Profile with two serial digital component cards and one analog composite input card, the serial digital inputs will be numbered 00-03 and the composite input will be numberd 04, **regardless** of where the composite card is placed.

| | |
|---|---|
| column 6: | optional separator (,) |

This is mandatory if column 7 is meaningful, optional otherwise.

| | |
|---|---|
| column 7: | optional parameters |

For inputs, this must be an S (for sharable). In addition, certain resources may specify additional parameters, starting in column 7. These parameters are read and assigned in the order given below.

| | |
|---|---|
| For JPEG codecs: | goalsize(bytes/field), minimum luminance Q, maximum luminance Q, minimum chrominance Q, and maximum chrominance Q. |
| For audio codecs: | input level, output level, window size |
| For timecode generators: | timecode format, timecode mode, time to seed, user bits to seed. |

Each resource allocation line is assigned an **index number** in sequential order independent of comment lines or crosspoint lines. Thus, the first resource line found in an NCS file is assigned index number 1, the second is assigned index number 2, etc.. These index numbers may be used by the crosspoint lines, which are described in the following section.

## Crosspoint lines

Crosspoint lines assign default and scheduled crosspoints between resources. A basic explanation of these events is included below. For a more detailed discussion of default and scheduled crosspoint events, see the *Profile SDK User Manual.*

The following is the column-by-column layout of crosspoint lines:

- column 1-2: source index number
- column 3: separator (-)
- column 4-5: destination index number
- column 6: separator (,)
- column 7: crosspoint indicator (‘C’ or ‘c’)
- column 8: ‘D’ or ‘S’ for default or schedule crosspoints

Scheduled crosspoint connections take precedence during playback, default crosspoints take effect at all other times (or in the absence of scheduled crosspoints). Typically, scheduled and default crosspoints are used in sets of three to define playback and E-to-E modes, as follows:

- default connection of an input to an output (E-E)
- default connection of an input to a codec (for recording)
- scheduled connection of the codec to the output (to override E-E during PB)

There are no audio crosspoints. A configuration file will only havecrosspoint lines for video and timecode.

The sample configuration file that follows provides examples of resource and crosspoint lines.

**Example 5. Sample configuration file**

```
02-00,S                 // index 1 - video input 0 shared
03-00                   // index 2 - video output 0
01-00,100000            // index 3 - video codec 0 at 100KB
01-02,CD                // Default connect video input->video output)
01-03,CD                // Default connect video input->video codec
03-02,CS                // Schedule connect video codec->video output

// now do the audio connections 1-4

00-00                   // index 4 - audio codecs
00-01                   // index 5 - audio codecs
00-02                   // index 6 - audio codecs
00-03                   // index 7 - audio codecs
10-00                   // index 8 - audio outputs 1-4
10-01                   // index 9
10-02                   // index 10
10-03                   // index 11

//  now do the LTC to recorder 1

04-00,S                 // index 12 - LTC input
07-00                   // index 13 - LTC output
09-00                   // index 14 - tcRecorder
12-13,CD
12-14,CD
14-13,CS
```

# System access commands (0x00)

System access commands are used to get the following Profile information:

- disk capacity
- software module version identification
- device identification
- file system name
- retrieving pending transaction exchanges
- audio, video, and timecode inputs and outputs
- VFS file access commands

### 00.00 Get Transaction Reply

(no API equivalent; communications only)

**Command:**

| | |
|---|---|
| data1: | Transaction number; the command returns the reply for the given transaction if it is available. |

**Reply:**

If the given transaction reply is ready, the transaction reply for the command that started the transaction is returned (see details for each command). If the given transaction reply is unavailable, the following is returned (full packet shown):

| | | |
|---|---|---|
| byte0: | 0x04 (Ack) | |
| byte1: | Bitmap, including one or more of the following: | |
| | bit0: | Command not implemented |
| | bit1: | Port busy |
| | bit2: | Incorrect number of bytes |
| | bit3: | Invalid transaction number |
| | bit4: | Invalid parameter |
| | bit5-6: | Reserved |
| | bit7: | RS422/RS232 event notification |
| byte2: | Transaction number from command data1 | |

### 00.01 Get Protocol Version

(no API equivalent; communications only)

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1: | Minor version number; a change in the minor version number indicates minor changes in the protocol |
| data2: | Major version number; a change in the major version number indicates major new features and can indicate incompatibility with controllers using lower major version numbers |

### 00.02 Get Device ID

(no API equivalent; communications only)

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1-n: | A null-terminated string indicating the device name. The string consists of the manufacturer's name, followed by a colon (":"), followed by the instrument name (e.g., "Tektronix:PDR100") |

### 00.03 Get Disk Capacity

(bitmask for calls)

**Command:**

At least one port must already be open for this call to function properly.

| | | | |
|---|---|---|---|
| data1-4: | Handle of port | | |
| data5: | Mask indicating capacity information needed: | | |
| | bit0: | Remaining capacity | (*VdrGetRecordCapacity*) |
| | bit1: | Total capacity | (*VdrGetTotalRecordCapacity*) |
| | bit2-7: | Reserved | |
| data6: | Drive number (0 to 2) | | |
| *OR* | | | |
| data6-n: | Name of dataset | | |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | |
|---|---|
| data1: | Transaction number |
| data2: | A copy of the request mask from command data5 (if the bit is not implemented it is cleared). If the given drive number or dataset name does not exist, this value will be set to zero and data3-n will not be returned. |
| data3-n: | If data2 is not zero, these bytes contain the values requested (value corresponding to the lowest bit in the mask first). |
| Remaining capacity: | Remaining capacity (integer), in fields. |
| Total capacity: | Total capacity (integer), in fields, when the disks have nothing recorded. |

### 00.04 Get Number of File Systems

(*CfgGetNumFileSystems*)

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1-4: | The number of file system partitions on the Profile (-1 indicates an error) |

If data1-4 indicates an error, data5-8 are also returned:

| | |
|---|---|
| data5-8: | Error code |

### 00.05 Get File System Name

(*CfgGetFileSystemName*)

| **Command:** | |
|---|---|
| data1-4: | File system number (starting at 0) |
| **Reply:** | |
| data1-n: | Name of file system with terminating null. (Null name indicates an error.) |
| If data1 is Null, data2-5 are also returned: | |
| data2-5: | Error code |

### 00.06 Get Number of Types

(bitmask for calls)

| **Command:** | | | |
|---|---|---|---|
| data1-2 | Mask of desired resource capabilities: | | |
| | bit0: | Inputs | (*CfgGetNumInputs*) |
| | bit1: | Codecs | (*CfgGetNumCodecs*) |
| | bit2: | Outputs | (*CfgGetNumOutputs*) |
| | bit3: | Timecode recorders | (*CfgGetNumTCRecorders*) |
| | bit4: | Timecode generators | (*CfgGetNumTCGenerators*) |
| | bit5-15: | Reserved | |
| data3-n: | If inputs, the media type: | | |
| | 0: | Audio | |
| | 1: | Video | |
| | 2: | LTC | |
| | 3: | VITC | |
| | If output, the media type: | | |
| | 0: | Audio | |
| | 1: | Video | |
| | 2: | LTC | |
| | 3: | VITC | |
| **Reply:** | | | |
| data1-2: | The bitmap mask after validation | | |
| data3-n: | The numbers of the resources requested in the bitmap (0xff indicates error) of the media types requested in the data 3-n fields | | |

### 00.07 Get Input Name

(*CfgGetInputName*)

| **Command:** | |
|---|---|
| data1: | Input type (audio=0, video=1, LTC=2, VITC=3) |
| data2: | Input number |
| **Reply:** | |
| data1-n: | Name of the desired type and number with terminating null (Null indicates an error) |
| If data1 is Null, data2-5 are also returned: | |
| data2-5: | Error code |

### 00.08 Get Output Name (*CfgGetOutputName*)

| **Command:** | |
|---|---|
| data1: | Output type (audio=0, video=1, LTC=2, VITC=3) |
| data2: | Output number |
| **Reply:** | |
| data1-n: | Name of the desired type and number with terminating null (Null indicates an error) |
| If data1 is Null, data2-5 are also returned: | |
| data2-5: | Error code |

### 00.09 Get Input Unit (*CfgGetInputUnitNum*)

| **Command:** | |
|---|---|
| data1: | Input type (audio=0, video=1, LTC=2, VITC=3) |
| data2-n: | Input name string |
| **Reply:** | |
| data1: | The associated input unit number (0xff indicates error) |

### 00.0A Get Output Unit (*CfgGetOutputUnitNum*)

| **Command:** | |
|---|---|
| data1: | Output type (audio=0, video=1, LTC=2, VITC=3) |
| data2-n: | Output name string |
| **Reply:** | |
| data1: | The associated output unit number (0xff indicates error) |

### 00.0B Get Standard (*CfgGetStandard*)

| **Command:** |
|---|
| (No data bytes required.) |
| **Reply:** |
| data1: The system standard (0=525, 1=625) |
| **Comments:** |
| This command gets the standard of the reference genlock card. All output playback will be formatted to this standard. Video codec format should also be to this standard. |

### 00.0C Get Video Standard (*CfgGetVideoInputStandard*)

| **Command:** | |
|---|---|
| data1: | Unit number (channel) |
| **Reply:** | |
| data1: | The video input's standard (0=525, 1=625; 0xff indicates error) |
| **Comments:** | |
| This is the format standard of a specific input. It allows an input of a different standard to be recorded. The codec must be set to the standard, but the system playback standard must be changed to play back the recorded material. | |

### 00.0D Get Current Directory *(VfsGetCurrentDirectory)*

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1-n: | Current directory path (Null indicates an error) |

If data1 is Null, data2-5 are also returned:

| | |
|---|---|
| data2-5: | Error code |

### 00.0E Set Current Directory *(VfsSetCurrentDirectory*

**Command:**

| | |
|---|---|
| data1-n: | Directory path to change to |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

### 00.0F Get Current Drive *(VfsGetCurrentDrive)*

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1-4: | The current drive number (-1 indicates an error) |

If data1-4 indicates an error, data5-8 are also returned:

| | |
|---|---|
| data5-8: | Error code |

### 00.10 Get TekVdr Major Version Number *(VdrGetMajorVersion)*

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1-4: | The major version number of the vdrport dll |

### 00.11 Get TekVdr Minor Version Number *(VdrGetMinorVersion)*

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1-4: | The minor version number of the vdrport dll |

### 00.12 Get Engine Major Version Number

(*VfsGetEngineMajorVersion*)

| **Command:** | | |
|---|---|---|
| (No data bytes required.) | | |
| **Reply:** | | |
| data1-4: | The major version number of the real-time engine subsystem | |

### 00.13 Get Engine Minor Version Number

(*VfsGetEngineMinorVersion*)

| **Command:** | | |
|---|---|---|
| data1-4: | port handle | (VdrHandle) |
| data5-8: | starting time code (as defined by SET_TIMECODE in *vrtypes.h*) | |
| data9-12: | user bits (same value striped throughout file) | |
| data13: | timecode format | (TcFormat) |
| **Reply:** | | |
| data1-4: | The minor version number of the real-time engine subsystem | |

### 00.14 Get Video Format

(*VdrGetVideoFormat*

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| **Reply:** | | |
| data1: | One of the following: | |
| | 0: | 525 line 60 Hz. |
| | 1: | 625 line 50 Hz. |

### 00.15 Set Video Format

(*VdrSetVideoFormat*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| data5: | One of the following: | |
| | 0: | 525 line 60 Hz. |
| | 1: | 625 line 50 Hz. |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 00.16 Get TekCfg Major Version Number

(*CfgGetMajorVersion*)

| **Command:** | |
|---|---|
| (No data bytes required.) | |
| **Reply:** | |
| data1-4: | The major version number of the configuration dll |

### 00.17 Get TekCfg Minor Version Number

(*CfgGetMinorVersion*)

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1-4: | The minor version number of the configuration dll |

### 00.18 Get Attributes by Type

(bitmask for calls)

**Command:**

| | | | |
|---|---|---|---|
| data1: | Desired resource type: | | |
| | 0: | Inputs | (*CfgGetInputAttributes*) |
| | 1: | Codecs | (*CfgGetCodecAttributes*) |
| | 2: | Outputs | (*CfgGetOutputAttributes*) |
| data2: | If resource type is inputs, the type | | |
| | 0: | for Audio | |
| | 1: | for Video | |
| | 2: | for LTC | |
| | 3: | for VITC | |
| | If codecs, the type | | |
| | 0: | for Audio | |
| | 1: | for Video | |
| | 2: | for Timecode | |
| | 3: | for MixEffects | |
| | If output, the type | | |
| | 0: | for Audio | |
| | 1: | for Video | |
| | 2: | for LTC | |
| | 3: | for VITC | |
| data3: | Logical unit number | | |
| data4: | Attribute value (see comments) | | |
| data5-8: | Second set of above data | | |
| data9-12: | Third set of above data | | |
| . . . | | | |
| data(n)-(n+3): | Last set of data | | |

**Reply:**

| | |
|---|---|
| data1-n: | The status (0-false, 1-true) of the attribute desired for the type specified. |

**Comments:**

Currently two audio attributes are supported:

- Level Adjustment = 100
- Equalization Adjustment = 101

Refer to the header file VDRCONF.H for additional supported attributes.

### 00.19 Get Video Field Count

(*VdrGetVideoFieldCount*)

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1-4: | Current system field count |

### 00.1A Wait For Video Field

(*VdrWaitForVideoField*)

**Command:**

| | | |
|---|---|---|
| data1: | Video field type on which to wait, as follows: | |
| | 0: | VideoField0246 |
| | 1: | VideoField1357 |
| | 2: | VideoField04 |
| | 3: | VideoField0 |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

### 00.1B Get TekPdr Major Version Number

(*PdrGetMajorVersion*)

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1-4: | The major version number of the PdrMovie dll |

### 00.1C Get TekPdr Minor Version Number

(*PdrGetMinorVersion*)

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1-4: | The minor version number of the PdrMovie dll |

### 00.1D Get TekVfs Major Version Number

(*VfsGetMajorVersion*)

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1-4: | The major version number of the PdrMovie dll |

### 00.1E Get TekVfs Minor Version Number

(*VfsGetMinorVersion*)

**Command:**

(No data bytes required.)

**Reply:**

| | |
|---|---|
| data1-4: | The minor version number of the PdrMovie dll |

### 00.1F Directory Exists

(*VfsDirectoryExists*)

**Command:**

| | |
|---|---|
| data1-n: | The directory name to determine its existence |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

### 00.20 Create Directory

(*VfsCreateDirectory*)

**Command:**

| | |
|---|---|
| data1-n: | The directory name to create |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

### 00.21 Vfs Get File Status

(bitmask for calls)

**Command:**

| | | | |
|---|---|---|---|
| data1-n: | The filename | | |
| data(n+1)-(n+2): | A mask indicating what status information is to be returned: | | |
| | bit0: | File attributes | (*VfsGetFileAttributes*) |
| | bit1: | File default mark-in | (*VfsGetFileDefaultMarkIn*) |
| | bit2: | File default mark-out | (*VfsGetFileDefaultMarkOut*) |
| | bit3: | File modification time | (*VfsGetFileModificationTime*) |
| | bit4: | File record duration | (*VfsGetFileRecordedDuration*) |
| | bit5: | File type | (*VfsGetFileType*) |
| | bit6: | File video format | (*VfsGetFileVideoFormat*) |
| | bit7-bit15: | Reserved | |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | | |
|---|---|---|---|
| data1: | Transaction number | | |
| data2-3: | A copy of the request mask from command data5-6 (unimplemented bits are cleared) | | |
| data4-n: | Status values requested (value corresponding to the lowest bit in the mask first): | | |
| | File attributes (4 bytes) | | See the Win32 SDK for breakdown |
| | File default mark-in (4 bytes) | | Mark-in for file |
| | File default mark-out (4 bytes) | | Mark-out for file |
| | File modification time (4 bytes) | | FILETIME format for last change to file |
| | File record duration (4 bytes) | | Length of file in fields |
| | File type (1 byte). One of the following types: | | |
| | 0: | FileTypeNonMedia | |
| | 1: | FileTypeVideo | |
| | 2: | FileTypeAudio | |
| | 3: | FileTypeTimecode | |
| | 4: | FileTypeMpegVideo | |
| | 5: | FileTypeDvcpro25Video | |
| | 6: | FileTypeDvcpro50Video | |
| | File video format (1 byte). One of the following: | | |
| | 0: | 525 line 60 Hz. | |
| | 1: | 625 line 50 Hz. | |

### 00.22 Vfs Create Media File

(*VfsCreateMediaFile*)

| **Command:** | | |
|---|---|---|
| data1-n: | Name of new media file null terminated | |
| data(n+1): | File type: | |
| | 1: | FileTypeVideo |
| | 2: | FileTypeAudio |
| | 3: | FileTypeTimecode |
| | 4: | FileTypeMpegVideo |
| | 5: | FileTypeDvcpro25Video |
| | 6: | FileTypeDvcpro50Video |
| data(n+2): | Video Format: | |
| | 0: | 525 line 60 Hz. |
| | 1: | 625 line 50 Hz. |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 00.23 Vfs Create File

(*VfsCreateFile*)

| **Command:** | | |
|---|---|---|
| data1-n: | Name of new file null terminated | |
| data(n+1)-(n+4): | Access | (See Win32 SDK for details) |
| data(n+5)-(n+8): | Share Mode | (See Win32 SDK for details) |
| data(n+9)-(n+12): | Create Flags | (See Win32 SDK for details) |
| data(n+13)-(n+16): | Attributes | (See Win32 SDK for details) |
| data(n+17)-(n+20): | Template Handle | (See Win32 SDK for details) |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2-5: | Handle of newly created file. (Null indicates error.) | |
| If data2-5 is Null: | | |
| data6-9: | Error code | |

### 00.24 Vfs Delete File

*(VfsDeleteFile)*

| **Command:** | | |
|---|---|---|
| data1-n: | Name of file null terminated | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 00.25 Vfs Move File

*(VfsMoveFile)*

| **Command:** | | |
|---|---|---|
| data1-n: | Name of file null terminated | |
| data(n+1)-(n+x): | New name of file null terminated | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 00.26 Vfs Read File

(*VfsReadFile*)

| **Command:** | | |
|---|---|---|
| data1-4: | File handle | |
| data5: | Number of bytes to read (up to maximum buffer size) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an OK status, data3-n are returned: | | |
| data3-n: | The actual data | |
| If data2 indicates an error, data3-6 are returned: | | |
| data3-6: | Error code | |

### 00.27 Vfs Set File Attributes

(*VfsSetFileAttributes*)

| **Command:** | | |
|---|---|---|
| data1-n: | Name of file null terminated | |
| data(n+1)-(n+4): | Attributes | (See Win32 SDK for details) |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 00.28 Vfs Write File

*(VfsWriteFile)*

| **Command:** | | |
|---|---|---|
| data1-4: | File handle | |
| data5: | Number of bytes to write (up to maximum buffer size) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| data3-6: | Number of bytes actually written | |
| If data2 indicates an error, data7-10 are also returned: | | |
| data7-10: | Error code | |

### 00.29 Vfs Erase Media Segment

*(VfsEraseMediaSegment)*

| **Command:** | | |
|---|---|---|
| data1-n: | File name | |
| data(n+1)-(n+4): | In-point | (Starting field number of erasure) |
| data(n+5)-(n+8): | Out-point | (Ending field number of erasure) |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 00.2A Vfs Close Handle

(*VfsCloseHandle*)

| **Command:** | | |
|---|---|---|
| data1-4: | File handle to close | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 00.2B Vfs Find First File

(*VfsFindFirstFile*)

| **Command:** | |
|---|---|
| data1-n: | Name on which to search |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Handle of operation. (Null indicates error.) |
| If data2-5 is not Null: | |
| data6-n: | WIN32_FIND_DATA structure (See Win32 SDK for details) |
| If data2-5 is Null: | |
| data6-9: | Error code |

### 00.2C Vfs Find Next File

(*VfsFindNextFile*)

| **Command:** | | |
|---|---|---|
| data1-n: | Handle of operation | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an OK status: | | |
| data3-n: | WIN32_FIND_DATA structure (See Win32 SDK for details) | |
| if data2 indicates an error: | | |
| data3-6: | Error code | |

### 00.2D Vfs Find Close

*(VfsFindClose)*

| **Command:** | |
|---|---|
| data1-4: | File operation handle (Returned by **VfsFindFirstFile** call) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |

### 00.2E Vfs Copy File

*(VfsCopyFile)*

| **Command:** | |
|---|---|
| data1-n: | File name |
| data(n+1)-(n+x): | New file name |
| data(n+x+1): | Wait for complete flag (Will not return until complete if set) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Handle of operation. (FF FF FF FF indicates error.) |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |

### 00.2F Vfs Copy Media Segment

*(VfsCopyMediaSegment)*

| **Command:** | |
|---|---|
| data1-n: | File name |
| data(n+1)-(n+x): | New file name |
| data(n+x+1)-(n+x+4): | First field number of source |
| data(n+x+5)-(n+x+8): | Last field number of source |
| data(n+x+9)-(n+x+12): | First field number of destination |
| data(n+x+13): | Wait for complete flag (Will not return until complete if set) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Handle of operation. (FF FF FF FF indicates error.) |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |

### 00.30 Vfs Move Media Segment

(*VfsMoveMediaSegment*)

| **Command:** | |
|---|---|
| data1-n: | File name |
| data(n+1)-(n+x): | New file name |
| data(n+x+1)-(n+x+4): | First field number of source |
| data(n+x+5)-(n+x+8): | Last field number of source |
| data(n+x+9)-(n+x+12): | First field number of destination |
| data(n+x+13): | Wait for complete flag (Will not return until complete if set) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Handle of operation. (FF FF FF FF indicates error.) |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |

### 00.31 Vfs Status of Copy

(*VfsStatusOfCopy*)

| **Command:** | | |
|---|---|---|
| data1-4: | Operation handle | |
| data5: | Wait for completion flag. (Will not return until complete if set.) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an OK status, data3-18 are returned: | | |
| data3-6: | Percent done | |
| data7-10: | Copy error | |
| data11-14: | Bytes transferred in last second | |
| data15-18: | Bytes per second transfer rate | |
| If data2 indicates an error, data3-6 are returned: | | |
| data3-6: | Error code | |

### 00.32 Vfs Cancel Copy

*(VfsCancelCopy)*

| | | |
|---|---|---|
| **Command:** | | |
| data1-4: | Operation handle (Returned by copy or move operation, prior 3 calls) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 00.33 Vfs Field Read

*(VfsFieldRead)*

| | | |
|---|---|---|
| **Command:** | | |
| data1-4: | File handle | |
| data5-8: | Field number | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an OK status, data3-10 are returned: | | |
| data3-6: | Memory pointer (returned from prior call + amount already transferred) | |
| data7-10: | Number of bytes actually read from field and ready to transfer | |
| If data2 indicates an error, data3-6 are returned: | | |
| data3-6: | Error code | |

### 00.34 Vfs Field Read Data

(*VfsFieldFree*)

| **Command:** | |
|---|---|
| data1-4: | Memory pointer (returned from prior call + amount already transferred) |
| data5-8: | Amount of data to transfer (up to maximum buffer size) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an OK status, data3-n are returned: | |
| data3-n: | The actual data |
| If data2 indicates an error, data3-6 are returned: | |
| data3-6: | Error code |

### 00.35 Vfs Field Free

(*VfsFieldFree*)

| **Command:** | |
|---|---|
| data1-4: | Memory pointer (returned from VfsFieldRead call) |
| **Reply:** | |
| data1: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data1 indicates an error, data2-5 are also returned: | |
| data2-5: | Error code |

### 00.36 Vfs Field Write Data

(*VfsFieldWrite*)

| **Command:** | |
|---|---|
| data1: | Field buffer number (0 for first time call) |
| data2-n: | Data to append to field (up to maximum buffer size) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Buffer number (0 is error) |

**Comment:**

The field buffer number must be 0 for a first time call as this indicates the desire for a new Field Write call to be started. A buffer will be allocated and the accompanying data saved. A return of 0 from any Vfs field write data call is an error. On subsequent calls, the buffer number is used to insure that the data is grouped together. The buffer number returned will be the same as the one passed in, or 0 for error.

### 00.37 Vfs Field Write Commit

*(VfsFieldWrite)*

| **Command:** | |
|---|---|
| data1-4: | File handle |
| data5-8: | Field number |
| data9: | Field buffer number (Returned from prior call) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |

### 00.38 Get Default Codec Timing

*(CfgGetCodecTimingMode)*

| **Command:** | |
|---|---|
| data1: | Codec type (PciCodecType) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Codec timing mode (VideoCodecTiming) |

### 00.39 Vfs Stripe Timecode

*(VfsStripeTimecode)*

| **Command:** | |
|---|---|
| data1-n: | File path to the timecode media file (dataset:/group/filename) |
| data(n+1)-(n+4): | Starting timecode (as defined by SET_TIMECODE in vdrtypes.h) |
| data(n+5)-(n+8): | User bits (same value striped throughout file) |
| data(n+9): | Timecode format (TcFormat in tstreamtypes.h) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an OK status, data3-6 are returned: | |
| data3-6: | Error code |

### 00.3A Vfs Set Default Codec Timing (disabled)

(no API equivalent)

| **Command:** | |
|---|---|
| data1: | 0 if systemtime; otherwise, localtime. |
| data2-5: | Time/date to set. (MS-DOS time/date format.) |
| **Reply:** | |
| data1: | 0 if successful; otherwise, error. |
| **Comments:** | |
| THIS COMMAND HAS BEEN DISABLED. | |

### 00.3B Vfs Set Time

(no API equivalent)

| **Command:** | |
|---|---|
| data1: | 0 if systemtime; otherwise, localtime. |
| data2-5: | Time/date to set. (MS-DOS time/date format.) |
| **Reply:** | |
| data1: | 0 if successful; otherwise, error. |

### 00.3C Vfs Get Time

(no API equivalent)

| **Command:** | |
|---|---|
| data1: | 0 if systemtime; otherwise, localtime. |
| data2-5: | Time/date to set. (MS-DOS time/date format.) |
| **Reply:** | |
| data1: | 0 if error; otherwise time/date value. |

### 00.3D Get Resource Name

(*CfgGetResourceName*)

*Engineering specification not yet available.*

### 00.3E Get Number of Resources

(*CfgGetNumResources*)

*Engineering specification not yet available.*

### 00.3F Vfs Query File System Space

(*VfsQueryFileSystemSpace*)

| **Command:** | | |
|---|---|---|
| data1-4: | File system number (starting at 0) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an OK status, data3-18 are returned: | | |
| data3-6: | Total space (high-order bytes) | |
| data7-10: | Total space (low-order bytes) | |
| data11-14: | Space remaining (high-order bytes) | |
| data15-18: | Space remaining (low-order bytes) | |
| If data2 indicates an error condition, data3-6 are returned: | | |
| data3-6: | Error code | |

### 00.40 Vfs Get File Time Code

(*VfsGetFileTimeCode*)

*Engineering specification not yet available.*

### 00.41 Vfs Get File Size

(*VfsGetFileSize*)

*Engineering specification not yet available.*

### 00.FF Event Poll

(*PdrCreateEvent*)

| **Command:** | |
|---|---|
| No command data. | |
| **Reply:** | |
| data1-4: | Handle of first triggered event (Null indicates error) |

# PLS serial protocol commands (0x0E)

PLS serial protocol commands are used to manage the Profile Library System.

## PLS serial protocol command structure

Command packets are documented as if they started with the first byte being the library server opcode. Returned data packets are also documented with the Profile protocol wrapper removed. This section does not describe the asynchronous transactions or basic error handling performed by the Profile protocol.

The constants that are common between the serial protocol and the API are defined in *plsconst.h*. Constants used exclusively by the serial protocol are defined in *pls_sp.h*. The header file *pls_sp.h* includes *plsconst.h*.

At the end of each command the length of command and response packets are given. The notation has a single number if the packet is of fixed length. If the packet is of variable size, the *<minimum>..<maximum>* value notation is used.

PLS serial commands consist of *definition fields* and *data types*.

### Definition fields

The following definition fields exist for each item in the protocol.

| | |
|---|---|
| description | This field is notes or descriptive information for the data item. |
| length | This is the length of the item in bytes. If the item is variable length, this field will contain two numbers that are the minimum and maximum allowed values. |
| offset | This is the offset from the first data byte that is not part of the Profile protocol wrapper. Data items that follow string locations are not fixed and the offset field contains "..." to indicate an unknown location value. |
| type | This field identifies the object's type (described in more detail below). |

### Serial protocol data types

The following data types are defined.

| | |
|---|---|
| action | This is an 8-bit variable that is a function action code. Action codes are defined as enums in the header files. |
| bitmap | This is a 16-bit variable that encodes command result control information. Command bitmaps describe what values are being requested in response to status, configuration and information commands. Response bitmaps describe what values are actually being supplied. When the bitmaps request incomplete responses, the unwanted variables are not returned and the offsets (displacements) listed in the text will not be correct. |
| bits | These are the bits that are part of the bitmaps. The value field shows the actual value. |
| bytes | A group of bytes used to reserve space. This is unused space. The field's size is noted in the length field and it is a constant. |
| const8 | The data item is an 8-bit constant. The value of the constant is given in the value field. |
| end | This denotes the end of a union. The offset field shows the largest offset that is needed to insure all possible parameter sets in a union can be transmitted. |

handle — This is a 32-bit protocol variable that encodes any of the library server handle types. It may be a library handle, a transport handle, a cartridge handle, etc..

long — A 32-bit signed integer variable.

origin — This is used in a union to move the offset back to the start of a union. The offset field will show the union's starting value.

schar — An 8-bit signed character or byte variable.

short — A 16-bit signed integer variable.

string — This is a variable size, null-terminated ASCII string. The minimum size is always one byte and the maximum size includes the null.

tm/dt — This is an MS-DOS time/date variable. It is converted to/from FILETIME function parameters as required. The MS-DOS *date* is a packed 16-bit value with the following format:

| Bits | Contents |
|---|---|
| 0 – 4 | Day of the month (1 – 31) |
| 5 – 8 | Nomth (1 = January, 2 = February, etc.) |
| 9 – 15 | Year fosteg from 1980 (add 1980 to get actual year) |

The MS-DOS *time* is a packed 16-bit value with the following format:

| Bits | Contents |
|---|---|
| 0 – 4 | Second divided by 2 |
| 5 – 10 | Minute (0 – 59) |
| 11 – 15 | Hour (0 – 23 on a 24-hour clock |

ubyte — An 8-bit unsigned character or byte variable.

ulong — A 32-bit unsigned integer variable.

union — This is the beginning of a set of fields that can overlay the same space in a response packet. They are not used in command packets. The offset field shows where a union begins.

ushort — A 16-bit unsigned integer variable.

### 0E.00 Create Extension

(no API equivalent)

| **Command:** | |
|---|---|
| data1-n: | name of extension |
| data1-7: | “tekpls” name of extension; Null-terminated string |
| data8: | 0x02 byte indication type of extension (EXT_TYPE_MACHINE) |
| **Reply:** | |
| Transaction response | |
| **Transaction Reply:** | |
| data1: | transaction number |
| data 2: | CMD2 value for the extension |
| (0 indicates error [data3-6: extended error code]) | |

### 0E.01 PlsGetMajorVersion

(*PlsGetMajorVersion*)

| **Command (packet size = 1):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x01 | Library opcode |
| **Returns (packet size = 6):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x01 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | short | | Major version number |

### 0E.02 PlsGetMinorVersion

(*PlsGetMinorVersion*)

| **Command (packet size = 1):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x02 | Library opcode |
| **Returns (packet size = 6):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x02 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | short | | Minor version number |

### 0E.03 PlsOpenLibrary

(*PlsOpenLibrary*)

| **Command (packet size = 7..55):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x03 | Library opcode |
| 1 | 2 | short | | Major version |
| 3 | 2 | short | | Minor version |
| 5 | 1..25 | string | | Library name |
| ... | 1..25 | string | | Session name |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x03 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | handle | | Library handle |

### 0E.04 PlsCloseLibrary

(*PlsCloseLibrary*)

| **Command (packet size = 5):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x04 | Library opcode |
| 1 | 4 | handle | | Library handle |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x04 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | handle | | Library handle |

### 0E.05 PlsGetLibraryConfig

(*PlsGetLibraryConfig*)

| **Command (packet size = 7):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x05 | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | Total bins in the library |
| | | bits | 0x0002 | Number of media mover robots |
| | | bits | 0x0004 | Number of common entry/exit ports |
| | | bits | 0x0008 | Number of cartridge entry ports |
| | | bits | 0x0010 | Number of cartridge exit ports |
| | | bits | 0x0020 | Number of tape transports |
| | | bits | 0x0040 | Library manufacture |
| | | bits | 0x0080 | Library model identification |
| | | bits | 0x0100 | Library system version |
| | | bits | 0x0200 | Library server software version |
| 3 | 4 | handle | | Library handle |
| **Returns (packet size = 6..150):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x05 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| | | | | Library configuration |
| (6) | 2 | short | | Total bins in the library |
| (8) | 2 | short | | Number of media mover robots |
| (10) | 2 | short | | Number of cartridge common entry/exit ports |
| (12) | 2 | short | | Number of cartridge entry ports |
| (14) | 2 | short | | Number of cartridge exit ports |
| (16) | 2 | short | | Number of tape transports |
| (18) | 1..33 | string | | Library manufacture |
| (51) | 1..33 | string | | Library model identification |
| (84) | 1..33 | string | | Library system version |
| (117) | 1..33 | string | | Library server software version |

### 0E.06 PlsGetLibraryStatus

(*PlsGetLibraryStatus*)

| Command (packet size = 7): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x06 | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | Status word |
| | | bits | 0x0002 | Number of empty bins |
| | | bits | 0x0004 | Number of bins with library tapes |
| | | bits | 0x0008 | Number of bins with cleaning tapes |
| | | bits | 0x0010 | Number of bins with other tapes |
| | | bits | 0x0020 | Number of available robots |
| | | bits | 0x0040 | Number of empty entry ports |
| | | bits | 0x0080 | Number of empty exit ports |
| | | bits | 0x0100 | Number of available transports |
| | | bits | 0x0200 | Device-specific information |
| 3 | 4 | handle | x | Library handle |

### 0E.06 PlsGetLibraryStatus (Continued)

(*PlsGetLibraryStatus*)

| Returns (packet size = 6..58): | | | | |
|---|---|---|---|---|
| 0 | 0 | ubyte | | Transaction number |
| 0 | 0 | const8 | 0x06 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| | | | | Library status |
| (6) | 4 | ulong | | Status word |
| (10) | 2 | short | | Number of empty bins |
| (12) | 2 | short | | Number of bins with library tapes |
| (14) | 2 | short | | Number of bins with cleaning tapes |
| (16) | 2 | short | | Number of bins with other tapes |
| (18) | 2 | short | | Number of available robots |
| (20) | 2 | short | | Number of empty entry ports |
| (22) | 2 | short | | Number of empty exit ports |
| (24) | 2 | short | | Number of available transports |
| >> (26) | | union | | Device-specific information |
| (26) | 32 | bytes | | Reserved space |
| ** (26) | | | origin | Exabyte 480 |
| (26) | 4 | long | | Total robot moves |
| (30) | 4 | long | | Pick and put retries |
| (34) | 4 | long | | Total scans |
| (38) | 4 | long | | Scan retries |
| (42) | 4 | long | | Scan failures |
| (46) | 4 | long | | Entry/exit port cycles |
| (50) | 8 | bytes | | Not used -- Zeros |
| ** (26) | | origin | | Ampex 412 |
| (26) | 32 | bytes | | Not used -- Zeros |
| << | | end | | Close union |

### 0E.07 PlsFindFirstTransportConfig

(*PlsFindFirstTransportConfig*)

| Command (packet size = 9): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x07 | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | The address for this transport |
| | | bits | 0x0002 | Location of transport in library |
| | | bits | 0x0004 | Name of host profile |
| | | bits | 0x0008 | Transport manufacture |
| | | bits | 0x0010 | Transport model identification |
| | | bits | 0x0020 | Transport version |

### 0E.07 PlsFindFirstTransportConfig

(*PlsFindFirstTransportConfig*)

| | | | | |
|---|---|---|---|---|
| 3 | 4 | handle | | Library handle |
| 7 | 2 | short | | Transport number |
| **Returns (packet size = 6..138):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x07 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| (6) | 4 | handle | | Transport configuration loop handle |
| | | | | Transport configuration |
| (10) | 2 | short | | The numeric address for this transport |
| (12) | 2 | short | | Location of transport in library |
| (14) | 1..25 | string | | Name of host profile |
| (39) | 1..33 | string | . | Transport manufacture |
| (72) | 1..33 | string | | Transport model identification |
| (105) | 1..33 | string | | Transport version |

### 0E.08 PlsFindNextTransportConfig

(*PlsFindNextTransportConfig*)

| **Command (packet size = 7):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x08 | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | The address for this transport |
| | | bits | 0x0002 | Location of transport in library |
| | | bits | 0x0004 | Name of host profile |
| | | bits | 0x0008 | Transport manufacture |
| | | bits | 0x0010 | Transport model identification |
| | | bits | 0x0020 | Transport version |
| 3 | 4 | handle | | Transport configuration loop handle |
| 7 | 2 | short | | Transport number |
| **Returns (packet size = 6..134):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x08 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| | | | | Transport configuration |
| (6) | 2 | short | | The numeric address for this transport |
| (8) | 2 | short | | Location of transport in library |
| (10) | 1..25 | string | | Name of host profile |
| (35) | 1..33 | string | . | Transport manufacture |
| (68) | 1..33 | string | | Transport model identification |
| (101) | 1..33 | string | | Transport version |

### 0E.09 PlsCloseTransportConfig

(*PlsCloseTransportConfig*)

| **Command (packet size = 5):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x09 | Library opcode |
| 1 | 4 | handle | | Transport configuration loop handle |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x09 | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.0A PlsFindFirstBinInfo

(*PlsFindFirstBinInfo*)

| Command (packet size = 10): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x0a | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | Status word |
| | | bits | 0x0002 | Bin number (location) |
| | | bits | 0x0004 | Cartridge label |
| | | bits | 0x0008 | Cartridge class this bin carries |
| 3 | 4 | handle | | Library handle |
| 7 | 1 | action | | Bin info action |
| 8 | 2 | short | | Starting bin |
| **Returns (packet size = 6..58):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x0a | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| (6) | 4 | handle | | Bin Information Loop Handle |
| | | | | Bin map information |
| (10) | 4 | ulong | | Status word |
| (14) | 2 | short | | Bin number (location) |
| (16) | 1..17 | string | | Cartridge label |
| (33) | 1..25 | string | | Cartridge class this bin carries |

### 0E.0B PlsFindNextBinInfo

(*PlsFindNextBinInfo*)

| **Command (packet size = 7):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x0b | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | Status word |
| | | bits | 0x0002 | Bin number (location) |
| | | bits | 0x0004 | Cartridge label |
| | | bits | 0x0008 | Cartridge class this bin carries |
| 3 | 4 | handle | | Bin Information Loop Handle |
| **Returns (packet size = 6..58):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x0b | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| | | | | Bin map information |
| (6) | 4 | ulong | | Status word |
| (10) | 2 | short | | Bin number (location) |
| (12) | 1..17 | string | | Cartridge label |
| (29) | 1..25 | string | | Cartridge class this bin carries |

### 0E.0C PlsCloseBinInfo

(*PlsCloseBinInfo*)

| **Command (packet size = 5):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x0c | Library opcode |
| 1 | 4 | handle | | Transport configuration loop handle |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x0c | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.0D PlsFindFirstCartridgeInfo

(*PlsFindFirstCartridgeInfo*)

| Command (packet size = 9..25): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x0d | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | Status word |
| | | bits | 0x0002 | Free space on cartridge (megabytes) |
| | | bits | 0x0004 | Largest free space (megabytes) |
| | | bits | 0x0008 | Partition size (megabytes) |
| | | bits | 0x0010 | Number of partitions |
| | | bits | 0x0020 | Barcode label for cartridge |
| | | bits | 0x0040 | First and next cart labels in MC set |
| | | bits | 0x0080 | If reserved, session name |
| | | bits | 0x0100 | Space allocation grouping control |
| | | bits | 0x0200 | Location hint for this cartridge |
| | | bits | 0x0400 | Load/unload cycles for cartridge |
| | | bits | 0x0800 | Tape motion time in seconds |
| | | bits | 0x1000 | Formatted, import/export and purge dates |
| 3 | 4 | handle | | Library handle |
| 7 | 1 | action | | Cartridge info action |
| 8 | 1..17 | string | | Starting cartridge |

**0E.0D PlsFindFirstCartridgeInfo (Continued)** (*PlsFindFirstCartridgeInfo*)

| Returns (packet size = 6..242): | | | | |
|---|---|---|---|---|
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x0d | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| (6) | 4 | handle | | Cartridge information loop handle |
| | | | | Cartridge status |
| (10) | 4 | ulong | | Status word |
| (14) | 4 | long | | Total free space on cartridge (megabytes) |
| (18) | 4 | long | | Largest free block on cartridge (megabytes) |
| (22) | 4 | long | | Partition size (megabytes) if partitioned |
| (26) | 2 | short | | Number of partitions or 0 if not partitioned |
| (28) | 4 | long | | Load/unload cycles for this cartridge |
| (32) | 4 | long | | Approximate tape motion time in seconds |
| (36) | 4 | tm/dt | | Last formatted, imported or exported |
| (40) | 1..17 | string | | Barcode label for cartridge |
| (57) | 1..17 | string | | For multicartridge set, first cartridge label |
| (74) | 1..17 | string | | For multicartridge set, next cartridge label |
| (91) | 1..25 | string | | If reserved, session name |
| (116) | 1..25 | string | | Space allocation grouping control |
| (141) | 1..101 | string | | If out of library, location hint |

### 0E.0E PlsFindNextCartridgeInfo

(*PlsFindNextCartridgeInfo*)

| **Command (packet size = 7):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x0e | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | A status word |
| | | bits | 0x0002 | Free space on cartridge (megabytes) |
| | | bits | 0x0004 | Largest free space (megabytes) |
| | | bits | 0x0008 | Partition size (megabytes) |
| | | bits | 0x0010 | Number of partitions |
| | | bits | 0x0020 | Barcode label for cartridge |
| | | bits | 0x0040 | First and next cart labels in MC set |
| | | bits | 0x0080 | If reserved, session name |
| | | bits | 0x0100 | Space allocation grouping control |
| | | bits | 0x0200 | Location hint for this cartridge |
| | | bits | 0x0400 | Load/unload cycles for cartridge |
| | | bits | 0x0800 | Tape motion time in seconds |
| | | bits | 0x1000 | Formatted, import/export and purge dates |
| 3 | 4 | handle | | Cartridge information loop handle |
| **Returns (packet size = 6..238):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x0e | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| | | | | Cartridge status |
| (6) | 4 | ulong | | Status word |
| (10) | 4 | long | | Total free space on cartridge (megabytes) |
| (14) | 4 | long | | Largest free block on cartridge (megabytes) |
| (18) | 4 | long | | Partition size (megabytes) if partitioned |
| (22) | 2 | short | | Number of partitions or 0 if not partitioned |
| (24) | 4 | long | | Load/unload cycles for this cartridge |
| (28) | 4 | long | | Approximate tape motion time in seconds |
| (32) | 4 | tm/dt | | Last formatted, imported or exported |
| (36) | 1..17 | string | | Barcode label for cartridge |
| (53) | 1..17 | string | | For multicartridge set, first cartridge label |
| (70) | 1..17 | string | | For multicartridge set, next cartridge label |
| (87) | 1..25 | string | | If reserved, session name |
| (112) | 1..25 | string | | Space allocation grouping control |
| (137) | 1..101 | string | | If out of library, location hint |

### 0E.0F PlsCloseCartridgeInfo

(*PlsCloseCartridgeInfo*)

| **Command (packet size = 5):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x0f | Library opcode |
| 1 | 4 | handle | | Cartridge information loop handle |
| **Returns (packet size = 6..238):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x0f | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.10 PlsFindFirstFileInfo

(*PlsFindFirstFileInfo*)

| **Command (packet size = 12..128):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x10 | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | A status word |
| | | bits | 0x0002 | Space required for the file (megabytes) |
| | | bits | 0x0004 | Unique name string (for future use) |
| | | bits | 0x0008 | File name (current name space) |
| | | bits | 0x0010 | File type (file contents) |
| | | bits | 0x0020 | Stored clip actual in point |
| | | bits | 0x0040 | Stored clip actual out point |
| | | bits | 0x0080 | Audio, video and time code track counts |
| | | bits | 0x0100 | File created and last used dates |
| | | bits | 0x0200 | Cartridge label for this file |
| | | bits | 0x0400 | Partition number file is in |
| 3 | 4 | handle | | Library handle |
| 7 | 1 | action | | File info action |
| 8 | 2 | short | | Partition number |
| 10 | 1..17 | string | | Cartridge label |
| ... | 1..101 | string | | File name |

### 0E.10 PlsFindFirstFileInfo (Continued)

(*PlsFindFirstFileInfo*)

| Returns (packet size = 6..222): | | | | |
|---|---|---|---|---|
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x10 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| (6) | 4 | handle | | File information loop handle |
| | | | | File information (directory entry) |
| (10) | 4 | ulong | | A status word |
| (14) | 4 | long | | Space required for the file (megabytes) |
| (18) | 4 | long | | Clip actual in point |
| (22) | 4 | long | | Clip actual out point |
| (26) | 2 | short | | Number of video tracks in this file |
| (28) | 2 | short | | Number of audio tracks |
| (30) | 2 | short | | Number of time code tracks |
| (32) | 4 | tm/dt | | When was the library file created |
| (36) | 4 | tm/dt | | When was the library file last referenced |
| (40) | 2 | short | | Partition number file is in |
| (42) | 4 | tm/dt | | Auto delete date for this clip |
| (46) | 1..33 | string | | Unique name string (for future use) |
| (79) | 1..101 | string | | File name (current name space) |
| (180) | 1..25 | string | | File type (file contents) |
| (205) | 1..17 | string | | Cartridge label for this file |

### 0E.11 PlsFindNextFileInfo

(*PlsFindNextFileInfo*)

| Command (packet size = 7): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x11 | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | A status word |
| | | bits | 0x0002 | Space required for the file (megabytes) |
| | | bits | 0x0004 | Unique name string (for future use) |
| | | bits | 0x0008 | File name (current name space) |
| | | bits | 0x0010 | File type (file contents) |
| | | bits | 0x0020 | Stored clip actual in point |
| | | bits | 0x0040 | Stored clip actual out point |
| | | bits | 0x0080 | Audio, video and time code track counts |
| | | bits | 0x0100 | File created and last used dates |
| | | bits | 0x0200 | Cartridge label for this file |
| | | bits | 0x0400 | Partition number file is in |
| 3 | 4 | handle | | File information loop handle |

### 0E.11 PlsFindNextFileInfo (Continued)

(*PlsFindNextFileInfo*)

| Returns (packet size = 6..218): | | | | |
|---|---|---|---|---|
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x11 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| | | | | File information (directory entry) |
| (6) | 4 | ulong | | A status word |
| (10) | 4 | long | | Space required for the file (megabytes) |
| (14) | 4 | long | | Clip actual in point |
| (18) | 4 | long | | Clip actual out point |
| (22) | 2 | short | | Number of video tracks in this file |
| (24) | 2 | short | | Number of audio tracks |
| (26) | 2 | short | | Number of time code tracks |
| (28) | 4 | tm/dt | | When was the library file created |
| (32) | 4 | tm/dt | | When was the library file last referenced |
| (36) | 2 | short | | Partition number file is in |
| (38) | 4 | tm/dt | | Auto delete date for this clip |
| (42) | 1..33 | string | | Unique name string (for future use) |
| (75) | 1..101 | string | | File name (current name space) |
| (176) | 1..25 | string | | File type (file contents) |
| (201) | 1..17 | string | | Cartridge label for this file |

### 0E.12 PlsCloseFileInfo

(*PlsCloseFileInfo*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x12 | Library opcode |
| 1 | 4 | handle | | File information loop handle |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x12 | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.13 PlsFindFirstHandle

(*PlsFindFirstHandle*)

| Command (packet size = 6): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x13 | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 1 | action | | Handle info action |
| **Returns (packet size = 13..133):** | | | | |
| 0 | 1 | ubyte | | Transaction number |

### 0E.13 PlsFindFirstHandle

(*PlsFindFirstHandle*)

| | | | | |
|---|---|---|---|---|
| 1 | 1 | const8 | 0x13 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Handle information loop handle |
| | | | | Handle information |
| 8 | 1 | ubyte | | Handle type |
| 9 | 4 | handle | | Handle |
| >> 13 | | union | | Library handle |
| 13 | 1..25 | string | | Session name |
| ** 13 | - | origin | | Cartridge handle |
| 13 | 1..17 | string | | Cartridge label |
| ** 13 | | origin | | Transport handle |
| 13 | 2 | short | | Transport Number |
| ** 13 | | origin | | File handle |
| 13 | 2 | short | | Partition number |
| 15 | 1..101 | string | | File name |
| ... | 1..17 | string | | Cartridge label |
| ** 13 | | origin | | Transaction handle |
| 13 | 4 | handle | | Library handle |
| 17 | 1 | ubyte | | Event tag |
| << | | end | | Close union |

### 0E.14 PlsFindNextHandle

(*PlsFindNextHandle*)

| **Command (packet size = 5):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x14 | Library opcode |
| 1 | 4 | handle | | Handle information loop handle |
| **Returns (packet size = 9..129):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x14 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| | | | | Handle information |
| 4 | 1 | ubyte | | Handle type |
| 5 | 4 | handle | | Handle |
| >> 9 | | union | | Library handle |
| 9 | 1..25 | string | | Session name |
| ** 9 | | origin | | Cartridge handle |
| 9 | 1..17 | string | | Cartridge label |
| ** 9 | | origin | | Transport handle |
| 9 | 2 | short | | Transport Number |
| ** 9 | | origin | | File handle |
| 9 | 2 | short | | Partition number |
| 11 | 1..101 | string | | File name |
| ... | 1..17 | string | | Cartridge label |
| ** 9 | | origin | | Transaction handle |
| 9 | 4 | handle | | Library handle |
| 13 | 1 | ubyte | | Event tag |
| << | | end | | Close union |

### 0E.15 PlsCloseHandle

(*PlsCloseFindHandle*)

| **Command (packet size = 5):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x15 | Library opcode |
| 1 | 4 | handle | | Handle information loop handle |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x15 | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.20 PlsConnectTransport

(*PlsConnectTransport*)

| **Command (packet size = 7):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x20 | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 2 | short | | Transport number |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x20 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transport handle |

### 0E.21 PlsAllocateTransport

(*PlsAllocateTransport*)

| **Command (packet size = 7):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x21 | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 2 | short | | Transport number |
| **Returns (packet size = 12):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x21 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |
| 8 | 4 | bytes | | Not used -- Zeros |

### 0E.22 PlsCloseTransport

(*PlsCloseTransport*)

| **Command (packet size = 5):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x22 | Library opcode |
| 1 | 4 | handle | | Transport |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x22 | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.23 PlsGetTransportStatus

(*PlsGetTransportStatus*)

| Command (packet size = 7): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x23 | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | A status word |
| | | bits | 0x0002 | The transport logical address |
| | | bits | 0x0004 | Who has reserved the transport |
| | | bits | 0x0008 | Barcode of cartridge if one loaded |
| | | bits | 0x0010 | Load/unload cycles for this transport |
| | | bits | 0x0020 | Tape motion time in seconds |
| | | bits | 0x0040 | Device specific data |
| 3 | 4 | handle | | Transport |

### 0E.23 PlsGetTransportStatus (Continued)

(*PlsGetTransportStatus*)

| Returns (packet size = 6..94): | | | | |
|---|---|---|---|---|
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x23 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| | | | | Transport status |
| (6) | 4 | ulong | | A status word |
| (10) | 2 | short | | The transport logical address |
| (12) | 4 | long | | Load/unload cycles for this transport |
| (16) | 4 | long | | Approximate tape motion time in seconds |
| (20) | 1..25 | string | | If reserved, session name |
| (45) | 1..17 | string | | Barcode of cartridge if one loaded |
| >> (62) | | union | | Device specific information |
| (62) | 32 | bytes | | Reserved space |
| ** | (62) | | origin | Exabyte Mammoth |
| (62) | 4 | long | | Recovered write errors |
| (66) | 4 | long | | Recovered read errors |
| (70) | 4 | long | | M-bytes written to tape |
| (74) | 4 | long | | M-bytes read |
| (78) | 2 | short | | Unrecovered write errors |
| (80) | 2 | short | | Unrecovered write errors |
| (82) | 12 | bytes | | Not used -- Zeros |
| ** (62) | | origin | | Ampex DST 312 |
| (62) | 4 | long | | Recovered write errors |
| (66) | 4 | long | | Recovered read errors |
| (70) | 2 | short | | Unrecovered write errors |
| (72) | 2 | short | | Unrecovered write errors |
| (74) | 20 | bytes | | Not used -- Zeros |
| << | | end | | Close union |

### 0E.24 PlsLoadTransport

(*PlsLoadTransport*)

| **Command (packet size = 9):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x24 | Library opcode |
| 1 | 4 | handle | | Transport |
| 5 | 4 | handle | | Cartridge |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x24 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.25 PlsUnloadTransport

(*PlsUnloadTransport*)

| **Command (packet size = 5):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x25 | Library opcode |
| 1 | 4 | handle | | Transport |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x25 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.26 PlsCleanTransport

(*PlsCleanTransport*)

| **Command (packet size = 9):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x26 | Library opcode |
| 1 | 4 | handle | | Transport |
| 5 | 4 | handle | | Cartridge |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x26 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.30 PlsConnectCartridge

(*PlsConnectCartridge*)

| Command (packet size = 6..22): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x30 | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 1..17 | string | | Cartridge label |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x30 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Cartridge handle |

### 0E.32 PlsAllocateCartridge

(*PlsAllocateCartridge*)

| Command (packet size = 6..22): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x32 | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 1..17 | string | | Cartridge label |
| **Returns (packet size = 12):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x32 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |
| 8 | 4 | bytes | | Not used -- Zeros |

### 0E.33 PlsCloseCartridge

(*PlsCloseCartridge*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x33 | Library opcode |
| 1 | 4 | handle | | Cartridge |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x33 | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.34 PlsGetCartridgeConfig

(*PlsGetCartridgeConfig*)

| **Command (packet size = 7):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x34 | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | Maximum possible space (megabytes) |
| | | bits | 0x0002 | Maximum possible partitions |
| | | bits | 0x0004 | Cartridge class identification string |
| | | bits | 0x0008 | Cartridge manufacture if known |
| | | bits | 0x0010 | The cartridge model (capacity, etc..) |
| 3 | 4 | handle | | Cartridge |
| **Returns (packet size = 6..103):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x34 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| | | | | Cartridge configuration |
| (6) | 4 | long | | Maximum possible space (megabytes) |
| (10) | 2 | short | | Maximum possible partitions |
| (12) | 1..25 | string | | Cartridge class identification string |
| (37) | 1..33 | string | | Cartridge manufacture if known |
| (70) | 1..33 | string | | The cartridge model (capacity, etc..) |

### 0E.35 PlsGetCartridgeStatus

(*PlsGetCartridgeStatus*)

| Command (packet size = 7): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x35 | Library opcode |
| 1 | 2 | bitmap | | Bitmap |
| | | bits | 0x0001 | A status word |
| | | bits | 0x0002 | Free space on cartridge (megabytes) |
| | | bits | 0x0004 | Largest free space (megabytes) |
| | | bits | 0x0008 | Partition size (megabytes) |
| | | bits | 0x0010 | Number of partitions |
| | | bits | 0x0020 | Barcode label for cartridge |
| | | bits | 0x0040 | First and next cart labels in MC set |
| | | bits | 0x0080 | If reserved, session name |
| | | bits | 0x0100 | Space allocation grouping control |
| | | bits | 0x0200 | Location hint for this cartridge |
| | | bits | 0x0400 | Load/unload cycles for cartridge |
| | | bits | 0x0800 | Tape motion time in seconds |
| | | bits | 0x1000 | Formatted, import/export and purge dates |
| 3 | 4 | handle | | Cartridge |
| **Returns (packet size = 6..238):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x35 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | bitmap | | Result bitmap |
| | | | | Cartridge status |
| (6) | 4 | ulong | | Status word |
| (10) | 4 | long | | Total free space on cartridge (megabytes) |
| (14) | 4 | long | | Largest free block on cartridge (megabytes) |
| (18) | 4 | long | | Partition size (megabytes) if partitioned |
| (22) | 2 | short | | Number of partitions or 0 if not partitioned |
| (24) | 4 | long | | Load/unload cycles for this cartridge |
| (28) | 4 | long | | Approximate tape motion time in seconds |
| (32) | 4 | tm/dt | | Last formatted, imported or exported |
| (36) | 1..17 | string | | Barcode label for cartridge |
| (53) | 1..17 | string | | For multicartridge set, first cartridge label |
| (70) | 1..17 | string | | For multicartridge set, next cartridge label |
| (87) | 1..25 | string | | If reserved, session name |
| (112) | 1..25 | string | | Space allocation grouping control |
| (137) | 1..101 | string | | If out of library, location hint |

### 0E.36 PlsInventoryCartridge

(*PlsInventoryCartridge*)

| **Command (packet size = 9):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x36 | Library opcode |
| 1 | 4 | handle | | Transport |
| 5 | 4 | handle | | Cartridge |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x36 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.37 PlsUpdateCartridge

(*PlsUpdateCartridge*)

| **Command (packet size = 9):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x37 | Library opcode |
| 1 | 4 | handle | | Transport |
| 5 | 4 | handle | | Cartridge |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x37 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.38 PlsFormatCartridge

(*PlsFormatCartridge*)

| **Command (packet size = 20..60):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x38 | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 4 | handle | | Transport |
| 9 | 1 | action | | Format action |
| 10 | 4 | long | | Master directory size (megabytes) |
| 14 | 4 | long | | Partition size (megabytes) |
| 18 | 1..17 | string | | Cartridge label |
| ... | 1..25 | string | | Category |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x38 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.3A PlsCopyCartridge

(*PlsCopyCartridge*)

| Command (packet size = 18): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x3a | Library opcode |
| 1 | 1 | action | | Copy action |
| 2 | 4 | handle | | Source transport |
| 6 | 4 | handle | | Source cartridge |
| 10 | 4 | handle | | Destination transport |
| 14 | 4 | handle | | Destination cartridge |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x3a | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.3C PlsExportCartridge

(*PlsExportCartridge*)

| Command (packet size = 7..107): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x3c | Library opcode |
| 1 | 4 | handle | | Cartridge |
| 5 | 1 | action | | Export action |
| 6 | 1..101 | string | | Location information |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x3c | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.3D PlsImportCartridge

(*PlsImportCartridge*)

| Command (packet size = 8..124): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x3d | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 1 | action | | Import action |
| 6 | 1..17 | string | | Cartridge label |
| ... | 1..101 | string | | Location information |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x3d | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.3E PlsGetLocationString

(*PlsGetLocationString*)

| **Command (packet size = 5):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x3e | Library opcode |
| 1 | 4 | handle | | Cartridge |
| **Returns (packet size = 5..105):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x3e | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 1..101 | string | | Location information |

### 0E.3F PlsSetLocationString

(*PlsSetLocationString*)

| **Command (packet size = 6..106):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x3f | Library opcode |
| 1 | 4 | handle | | Cartridge |
| 5 | 1..101 | string | | Location information |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x3f | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.40 PlsSetCartDescription

(*PlsSetCartDescription*)

| **Command (packet size = 6..106):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x40 | Library opcode |
| 1 | 4 | handle | | Cartridge |
| 5 | 1..101 | string | | Cartridge description |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x40 | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.41 PlsGetCartDescription

(*PlsGetCartDescription*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x41 | Library opcode |
| 1 | 4 | handle | | Cartridge |
| **Returns (packet size = 5..105):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x41 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 1..101 | string | | Cartridge description |

### 0E.42 PlsImportLoadCartridge

(*PlsImportLoadCartridge*)

| Command (packet size = 12..128): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x42 | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 4 | handle | | Transport handle |
| 9 | 1 | action | | Import action |
| 10 | 1..17 | string | | Cartridge label |
| ... | 1..101 | string | | Location information |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x42 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.43 PlsGetPartitionMap

(*PlsGetPartitionMap*)

| Command (packet size = 9): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x43 | Library opcode |
| 1 | 4 | handle | | Cartridge |
| 5 | 2 | short | | Starting partition number |
| 7 | 2 | short | | Requested number of partitions |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x43 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.50 PlsConnectFile

(*PlsConnectFile*)

| Command (packet size = 12..112): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x50 | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 4 | handle | | Cartridge |
| 9 | 2 | short | | Partition number |
| 11 | 1..101 | string | | File name |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x50 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | File handle |

### 0E.51 PlsCloseFile

(*PlsCloseFile*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x51 | Library opcode |
| 1 | 4 | handle | | File |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x51 | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.53 PlsArchiveClip

(*PlsArchiveClip*)

| Command (packet size = 17): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x53 | Library opcode |
| 1 | 4 | handle | | File |
| 5 | 4 | long | | Archive in point |
| 9 | 4 | long | | Archive out point |
| 13 | 4 | handle | | Transport |
| **Returns (packet size = 11):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x53 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |
| 8 | 3 | bytes | | Not used -- Zeros |

### 0E.54 PlsArchiveDataFile

(*PlsArchiveDataFile*)

| Command (packet size = 10..110): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x54 | Library opcode |
| 1 | 4 | handle | | File |
| 5 | 4 | handle | | Transport |
| 9 | 1..101 | string | | Source Win NT data file name |
| **Returns (packet size = 11):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x54 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |
| 8 | 3 | bytes | | Not used -- Zeros |

### 0E.56 PlsRestore

(*PlsRestore*)

| Command (packet size = 17): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x56 | Library opcode |
| 1 | 4 | handle | | File |
| 5 | 4 | long | | Restore in point |
| 9 | 4 | long | | Restore out point |
| 13 | 4 | handle | | Transport |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x56 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.57 PlsRenameFile

(*PlsRenameFile*)

| Command (packet size = 6..106): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x57 | Library opcode |
| 1 | 4 | handle | | File |
| 5 | 1..101 | string | | New file name |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x57 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.58 PlsDeleteFile

(*PlsDeleteFile*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x58 | Library opcode |
| 1 | 4 | handle | | File |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x58 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.5A PlsSetFileDescription

(*PlsSetFileDescription*)

| Command (packet size = 6..56): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x5a | Library opcode |
| 1 | 4 | handle | | File |
| 5 | 1..51 | string | | File description |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x5a | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.5B PlsGetFileDescription

(*PlsGetFileDescription*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x5b | Library opcode |
| 1 | 4 | handle | | File |
| **Returns (packet size = 5..55):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x5b | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 1..51 | string | | File description |

### 0E.5F PlsRestoreDataFile

(*PlsRestoreDataFile*)

| **Command (packet size = 10..110):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x5f | Library opcode |
| 1 | 4 | handle | | File |
| 5 | 4 | handle | | Transport |
| 9 | 1..101 | string | | Destination Win NT file name path |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x5f | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.60 PlsSetBackupDirCommand

(*PlsSetBackupDir*)

| **Command (packet size = 7..79):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x60 | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 1..65 | string | | Backup directory path |
| ... | 1..9 | string | | Backup drive |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x60 | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.61 PlsGetBackupDirCommand

(*PlsGetBackupDir*)

| **Command (packet size = 5):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x61 | Library opcode |
| 1 | 4 | handle | | Library handle |
| **Returns (packet size = 6..78):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x61 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 1..65 | string | | Backup directory path |
| ... | 1..9 | string | | Backup drive |

### 0E.62 PlsBackupCatalogCommand

(*PlsBackupCatalog*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x62 | Library opcode |
| 1 | 4 | handle | | Library handle |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x62 | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.63 PlsGetClipSize

(*PlsGetClipSize*)

| Command (packet size = 17): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x63 | Library opcode |
| 1 | 4 | handle | | File |
| 5 | 4 | long | | In point |
| 9 | 4 | long | | Out point |
| 13 | 4 | handle | | Transport handle |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x63 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.70 PlsSetEventMask

(*PlsSetEventMask*)

| Command (packet size = 7): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x70 | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 2 | ushort | | Event mask |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x70 | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.71 PlsGetEventMask

(*PlsGetEventMask*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x71 | Library opcode |
| 1 | 4 | handle | | Library handle |
| **Returns (packet size = 6):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x71 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 2 | ushort | | Event mask |

### 0E.72 PlsGetAnyEvent

(*PlsGetAnyEvent*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x72 | Library opcode |
| 1 | 4 | handle | | Library handle |
| **Returns (packet size = 5..245):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x72 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| | | | | Event result information |
| 4 | 1 | ubyte | | Event tag |
| >> 5 | | union | | Event information |
| ** 5 | | origin | | Concurrent command completion |
| 5 | 4 | handle | | Transaction handle |
| 9 | 2 | short | | Command return code |
| >> 11 | | union | | Command completion results |
| ** 11 | | origin | | PlsEventCmdAllocateTransport |
| 11 | 4 | handle | | Transport handle |
| ** 11 | | origin | | PlsEventCmdAllocateCartridge |
| 11 | 4 | handle | | Cartridge handle |
| ** 11 | | origin | | PlsGetPartMap |
| 11 | 2 | short | | Actual number of entries returned |

**0E.72 PlsGetAnyEvent (Continued)** (*PlsGetAnyEvent*)

| | | | |
|---|---|---|---|
| | | | Zero or more entries as follows: |
| (13) | 4 | long | 1st partition free space |
| (17) | 4 | long | 2nd partition free space |
| (21) | 4 | long | 3rd partition free space |
| (25) | 4 | long | 4th partition free space |
| (29) | 4 | long | 5th partition free space |
| (33) | 4 | long | 6th partition free space |
| (37) | 4 | long | 7th partition free space |
| (41) | 4 | long | 8th partition free space |
| (45) | 4 | long | 9th partition free space |
| (49) | 4 | long | 10th partition free space |
| (53) | 4 | long | 11th partition free space |
| (57) | 4 | long | 12th partition free space |
| (61) | 4 | long | 13th partition free space |
| (65) | 4 | long | 14th partition free space |
| (69) | 4 | long | 15th partition free space |
| (73) | 4 | long | 16th partition free space |
| (77) | 4 | long | 17th partition free space |
| (81) | 4 | long | 18th partition free space |
| (85) | 4 | long | 19th partition free space |
| (89) | 4 | long | 20th partition free space |
| (93) | 4 | long | 21th partition free space |
| (97) | 4 | long | 22th partition free space |
| (101) | 4 | long | 23th partition free space |
| (105) | 4 | long | 24th partition free space |
| (109) | 4 | long | 25th partition free space |
| (113) | 4 | long | 26th partition free space |
| (117) | 4 | long | 27th partition free space |
| (121) | 4 | long | 28th partition free space |
| (125) | 4 | long | 29th partition free space |
| (129) | 4 | long | 30th partition free space |
| (133) | 4 | long | 31th partition free space |
| (137) | 4 | long | 32th partition free space |
| (141) | 4 | long | 33th partition free space |
| (145) | 4 | long | 34th partition free space |
| (149) | 4 | long | 35th partition free space |
| (153) | 4 | long | 36th partition free space |
| (157) | 4 | long | 37th partition free space |
| (161) | 4 | long | 38th partition free space |
| (165) | 4 | long | 39th partition free space |
| (169) | 4 | long | 40th partition free space |

**0E.72 PlsGetAnyEvent (Continued)** (*PlsGetAnyEvent*)

| | | | |
|---|---|---|---|
| (173) | 4 | long | 41th partition free space |
| (177) | 4 | long | 42th partition free space |
| (181) | 4 | long | 43th partition free space |
| (185) | 4 | long | 44th partition free space |
| (189) | 4 | long | 45th partition free space |
| (193) | 4 | long | 46th partition free space |
| (197) | 4 | long | 47th partition free space |
| (201) | 4 | long | 48th partition free space |
| (205) | 4 | long | 49th partition free space |
| (209) | 4 | long | 50th partition free space |
| (213) | 4 | long | 51th partition free space |
| (217) | 4 | long | 52th partition free space |
| (221) | 4 | long | 53th partition free space |
| (225) | 4 | long | 54th partition free space |
| (229) | 4 | long | 55th partition free space |
| (233) | 4 | long | 56th partition free space |
| (237) | 4 | long | 57th partition free space |
| (241) | 4 | long | 58th partition free space |
| ** (11) | | origin | ArchiveClip, ArchiveDataFile, ArchiveSet |

**0E.72 PlsGetAnyEvent (Continued)** (*PlsGetAnyEvent*)

| | | | |
|---|---|---|---|
| | | | File stored result (where it was placed) |
| (11) | 2 | short | Partition number |
| (13) | 1..17 | string | Cartridge label |
| ** | (11) | origin | GetClipSize |
| (11) | 4 | long | File size |
| ** | (11) | origin | Extended completion for import and export |
| (11) | 1..17 | string | Cartridge label |
| ** | (11) | origin | Extended completion for format |
| (11) | 4 | long | Partition size (megabytes) |
| (15) | 2 | short | Number of partitions |
| (17) | 1..17 | string | Cartridge label |
| << | | end | Close union |
| ** (5) | | origin | Import, Export and Delete all files |
| (5) | 1..17 | string | Cartridge label |
| ** (5) | | origin | PlsEventNewFile |
| (5) | 2 | short | Partition number |
| (7) | 1..17 | string | Cartridge label |
| (24) | 1..101 | string | File name |
| ** (5) | | origin | PlsEventDeleteFile |
| (5) | 2 | short | Partition number |
| (7) | 1..17 | string | Cartridge label |
| (24) | 1..101 | string | File name |
| ** (5) | | origin | PlsEventRenameFile |
| (5) | 2 | short | Partition number |
| (7) | 1..17 | string | Cartridge label |
| (24) | 1..101 | string | File name |
| (125) | 1..101 | string | New file name |
| ** (5) | | origin | PlsEventTransOnline & PlsEventTransOffline |
| (5) | 2 | short | Transport number |
| << | | end | Close union |

### 0E.73 PlsGetAsynchEvent

(*PlsGetAsynchEvent*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x73 | Library opcode |
| 1 | 4 | handle | | Library handle |
| **Returns (packet size = 5..245):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x73 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| | | | | Event result information |
| 4 | 1 | ubyte | | Event tag |
| >> 5 | | union | | Event information |
| ** 5 | | origin | | Concurrent command completion |
| 5 | 4 | handle | | Transaction handle |
| 9 | 2 | short | | Command return code |
| >> 11 | | union | | Command completion results |
| ** 11 | | origin | | PlsEventCmdAllocateTransport |
| 11 | 4 | handle | | Transport handle |
| ** 11 | | origin | | PlsEventCmdAllocateCartridge |
| 11 | 4 | handle | | Cartridge handle |
| ** 11 | | origin | | PlsGetPartMap |
| 11 | 2 | short | | Actual number of entries returned |

**0E.73 PlsGetAsynchEvent (Continued)** (*PlsGetAsynchEvent*)

| | | | |
|---|---|---|---|
| | | | Zero or more entries as follows: |
| (13) | 4 | long | 1st partition free space |
| (17) | 4 | long | 2nd partition free space |
| (21) | 4 | long | 3rd partition free space |
| (25) | 4 | long | 4th partition free space |
| (29) | 4 | long | 5th partition free space |
| (33) | 4 | long | 6th partition free space |
| (37) | 4 | long | 7th partition free space |
| (41) | 4 | long | 8th partition free space |
| (45) | 4 | long | 9th partition free space |
| (49) | 4 | long | 10th partition free space |
| (53) | 4 | long | 11th partition free space |
| (57) | 4 | long | 12th partition free space |
| (61) | 4 | long | 13th partition free space |
| (65) | 4 | long | 14th partition free space |
| (69) | 4 | long | 15th partition free space |
| (73) | 4 | long | 16th partition free space |
| (77) | 4 | long | 17th partition free space |
| (81) | 4 | long | 18th partition free space |
| (85) | 4 | long | 19th partition free space |
| (89) | 4 | long | 20th partition free space |
| (93) | 4 | long | 21th partition free space |
| (97) | 4 | long | 22th partition free space |
| (101) | 4 | long | 23th partition free space |
| (105) | 4 | long | 24th partition free space |
| (109) | 4 | long | 25th partition free space |
| (113) | 4 | long | 26th partition free space |
| (117) | 4 | long | 27th partition free space |
| (121) | 4 | long | 28th partition free space |
| (125) | 4 | long | 29th partition free space |
| (129) | 4 | long | 30th partition free space |
| (133) | 4 | long | 31th partition free space |
| (137) | 4 | long | 32th partition free space |
| (141) | 4 | long | 33th partition free space |
| (145) | 4 | long | 34th partition free space |
| (149) | 4 | long | 35th partition free space |
| (153) | 4 | long | 36th partition free space |
| (157) | 4 | long | 37th partition free space |
| (161) | 4 | long | 38th partition free space |
| (165) | 4 | long | 39th partition free space |
| (169) | 4 | long | 40th partition free space |

**0E.73 PlsGetAsynchEvent (Continued)** (*PlsGetAsynchEvent*)

| | | | |
|---|---|---|---|
| (173) | 4 | long | 41th partition free space |
| (177) | 4 | long | 42th partition free space |
| (181) | 4 | long | 43th partition free space |
| (185) | 4 | long | 44th partition free space |
| (189) | 4 | long | 45th partition free space |
| (193) | 4 | long | 46th partition free space |
| (197) | 4 | long | 47th partition free space |
| (201) | 4 | long | 48th partition free space |
| (205) | 4 | long | 49th partition free space |
| (209) | 4 | long | 50th partition free space |
| (213) | 4 | long | 51th partition free space |
| (217) | 4 | long | 52th partition free space |
| (221) | 4 | long | 53th partition free space |
| (225) | 4 | long | 54th partition free space |
| (229) | 4 | long | 55th partition free space |
| (233) | 4 | long | 56th partition free space |
| (237) | 4 | long | 57th partition free space |
| (241) | 4 | long | 58th partition free space |
| ** (11) | | origin | ArchiveClip, ArchiveDataFile, ArchiveSet |

### 0E.73 PlsGetAsynchEvent (Continued)

(*PlsGetAsynchEvent*)

| | | | |
|---|---|---|---|
| | | | File stored result (where it was placed) |
| (11) | 2 | short | Partition number |
| (13) | 1..17 | string | Cartridge label |
| ** (11) | | origin | GetClipSize |
| (11) | 4 | long | File size |
| ** (11) | | origin | Extended completion for import and export |
| (11) | 1..17 | string | Cartridge label |
| ** (11) | | origin | Extended completion for format |
| (11) | 4 | long | Partition size (megabytes) |
| (15) | 2 | short | Number of partitions |
| (17) | 1..17 | string | Cartridge label |
| << | | end | Close union |
| ** (5) | | origin | Import, Export and Delete all files |
| (5) | 1..17 | string | Cartridge label |
| ** (5) | | origin | PlsEventNewFile |
| (5) | 2 | short | Partition number |
| (7) | 1..17 | string | Cartridge label |
| (24) | 1..101 | string | File name |
| ** (5) | | origin | PlsEventDeleteFile |
| (5) | 2 | short | Partition number |
| (7) | 1..17 | string | Cartridge label |
| (24) | 1..101 | string | File name |
| ** (5) | | origin | PlsEventRenameFile |
| (5) | 2 | short | Partition number |
| (7) | 1..17 | string | Cartridge label |
| (24) | 1..101 | string | File name |
| (125) | 1..101 | string | New file name |
| ** (5) | | origin | PlsEventTransOnline & PlsEventTransOffline |
| (5) | 2 | short | Transport number |
| << | | end | Close union |

### 0E.74 PlsGetCommandEvent

(*PlsGetCommandEvent*)

| Command (packet size = 9): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x74 | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 4 | handle | | Transaction handle |
| **Returns (packet size = 5..245):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x74 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| | | | | Event result information |
| 4 | 1 | ubyte | | Event tag |
| >> 5 | | union | | Event information |
| ** 5 | | origin | | Concurrent command completion |
| 5 | 4 | handle | | Transaction handle |
| 9 | 2 | short | | Command return code |
| >> 11 | | union | | Command completion results |
| ** 11 | | origin | | PlsEventCmdAllocateTransport |
| 11 | 4 | handle | | Transport handle |
| ** 11 | | origin | | PlsEventCmdAllocateCartridge |
| 11 | 4 | handle | | Cartridge handle |
| ** 11 | | origin | | PlsGetPartMap |
| 11 | 2 | short | | Actual number of entries returned |

**0E.74 PlsGetCommandEvent (Continued)** (*PlsGetCommandEvent*)

| | | | |
|---|---|---|---|
| | | | Zero or more entries as follows: |
| (13) | 4 | long | 1st partition free space |
| (17) | 4 | long | 2nd partition free space |
| (21) | 4 | long | 3rd partition free space |
| (25) | 4 | long | 4th partition free space |
| (29) | 4 | long | 5th partition free space |
| (33) | 4 | long | 6th partition free space |
| (37) | 4 | long | 7th partition free space |
| (41) | 4 | long | 8th partition free space |
| (45) | 4 | long | 9th partition free space |
| (49) | 4 | long | 10th partition free space |
| (53) | 4 | long | 11th partition free space |
| (57) | 4 | long | 12th partition free space |
| (61) | 4 | long | 13th partition free space |
| (65) | 4 | long | 14th partition free space |
| (69) | 4 | long | 15th partition free space |
| (73) | 4 | long | 16th partition free space |
| (77) | 4 | long | 17th partition free space |
| (81) | 4 | long | 18th partition free space |
| (85) | 4 | long | 19th partition free space |
| (89) | 4 | long | 20th partition free space |
| (93) | 4 | long | 21th partition free space |
| (97) | 4 | long | 22th partition free space |
| (101) | 4 | long | 23th partition free space |
| (105) | 4 | long | 24th partition free space |
| (109) | 4 | long | 25th partition free space |
| (113) | 4 | long | 26th partition free space |
| (117) | 4 | long | 27th partition free space |
| (121) | 4 | long | 28th partition free space |
| (125) | 4 | long | 29th partition free space |
| (129) | 4 | long | 30th partition free space |
| (133) | 4 | long | 31th partition free space |
| (137) | 4 | long | 32th partition free space |
| (141) | 4 | long | 33th partition free space |
| (145) | 4 | long | 34th partition free space |
| (149) | 4 | long | 35th partition free space |
| (153) | 4 | long | 36th partition free space |
| (157) | 4 | long | 37th partition free space |
| (161) | 4 | long | 38th partition free space |
| (165) | 4 | long | 39th partition free space |
| (169) | 4 | long | 40th partition free space |

**0E.74 PlsGetCommandEvent (Continued)** (*PlsGetCommandEvent*)

| | | | |
|---|---|---|---|
| (173) | 4 | long | 41th partition free space |
| (177) | 4 | long | 42th partition free space |
| (181) | 4 | long | 43th partition free space |
| (185) | 4 | long | 44th partition free space |
| (189) | 4 | long | 45th partition free space |
| (193) | 4 | long | 46th partition free space |
| (197) | 4 | long | 47th partition free space |
| (201) | 4 | long | 48th partition free space |
| (205) | 4 | long | 49th partition free space |
| (209) | 4 | long | 50th partition free space |
| (213) | 4 | long | 51th partition free space |
| (217) | 4 | long | 52th partition free space |
| (221) | 4 | long | 53th partition free space |
| (225) | 4 | long | 54th partition free space |
| (229) | 4 | long | 55th partition free space |
| (233) | 4 | long | 56th partition free space |
| (237) | 4 | long | 57th partition free space |
| (241) | 4 | long | 58th partition free space |
| ** (11) | | origin | ArchiveClip, ArchiveDataFile, ArchiveSet |

**0E.74 PlsGetCommandEvent (Continued)** (*PlsGetCommandEvent*)

| | | | |
|---|---|---|---|
| | | | File stored result (where it was placed) |
| (11) | 2 | short | Partition number |
| (13) | 1..17 | string | Cartridge label |
| ** (11) | | origin | GetClipSize |
| (11) | 4 | long | File size |
| ** (11) | | origin | Extended completion for import and export |
| (11) | 1..17 | string | Cartridge label |
| ** (11) | | origin | Extended completion for format |
| (11) | 4 | long | Partition size (megabytes) |
| (15) | 2 | short | Number of partitions |
| (17) | 1..17 | string | Cartridge label |
| << | | end | Close union |
| ** (5) | | origin | Import, Export and Delete all files |
| (5) | 1..17 | string | Cartridge label |
| ** (5) | | origin | PlsEventNewFile |
| (5) | 2 | short | Partition number |
| (7) | 1..17 | string | Cartridge label |
| (24) | 1..101 | string | File name |
| ** (5) | | origin | PlsEventDeleteFile |
| (5) | 2 | short | Partition number |
| (7) | 1..17 | string | Cartridge label |
| (24) | 1..101 | string | File name |
| ** (5) | | origin | PlsEventRenameFile |
| (5) | 2 | short | Partition number |
| (7) | 1..17 | string | Cartridge label |
| (24) | 1..101 | string | File name |
| (125) | 1..101 | string | New file name |
| ** (5) | | origin | PlsEventTransOnline & PlsEventTransOffline |
| (5) | 2 | short | Transport number |
| << | | end | Close union |

### 0E.75 PlsSetModes

(*PlsSetModes*)

| Command (packet size = 37): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x75 | Library opcode |
| 1 | 4 | handle | | Library handle |
| | | | | Library server modes |
| 5 | 4 | ulong | | Single bit mode flags |
| 9 | 28 | bytes | | Reserved for future use |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x75 | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.76 PlsGetModes

(*PlsGetModes*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x76 | Library opcode |
| 1 | 4 | handle | | Library handle |
| **Returns (packet size = 36):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x76 | Library opcode |
| 2 | 2 | ushort | - | Library error code |
| | | | | Library server modes |
| 4 | 4 | ulong | - | Single bit mode flags |
| 8 | 28 | bytes | - | Reserved for future use |

### 0E.77 PlsSetPath

(*PlsSetPath*)

| Command (packet size = 6..106): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x77 | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 1..101 | string | | Path |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x77 | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.78 PlsGetPath

(*PlsGetPath*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x78 | Library opcode |
| 1 | 4 | handle | | Library handle |
| **Returns (packet size = 5..105):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x78 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 1..101 | string | | Path |

### 0E.79 PlsGetTimeDate

(*PlsGetTimeDate*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x79 | Library opcode |
| 1 | 4 | handle | | Library handle |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x79 | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | tm/dt | | Current library server time/date |

### 0E.7A PlsAddTransport

(*PlsAddTransport*)

| Command (packet size = 11..59): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x7a | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 2 | short | | Transport number |
| 7 | 2 | short | | Robot address |
| 9 | 1..25 | string | | Host name |
| ... | 1..25 | string | | Device address |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x7a | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.7B PlsRemoveTransport

(*PlsRemoveTransport*)

| Command (packet size = 7): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x7b | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 2 | short | | Transport number |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x7b | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.7C PlsInventoryLibrary

(*PlsInventoryLibrary*)

| Command (packet size = 5): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x7c | Library opcode |
| 1 | 4 | handle | | Library handle |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x7c | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.7D PlsGetStatusCommand

(*PlsGetStatusCommand*)

| Command (packet size = 9): | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x7d | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 4 | handle | | Transaction handle |
| **Returns (packet size = 9):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x7d | Library opcode |
| 2 | 2 | ushort | | Library error code |
| | | | | Command status result |
| 4 | 4 | ulong | | Status bits |
| 8 | 1 | schar | | Percent complete (0..100) |

### 0E.7E PlsCancelCommand

(*PlsCancelCommand*)

| **Command (packet size = 9):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x7e | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 4 | handle | | Transaction handle |
| **Returns (packet size = 4):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x7e | Library opcode |
| 2 | 2 | ushort | | Library error code |

### 0E.7F PlsHouseKeeping

(*PlsHouseKeeping*)

| **Command (packet size = 10..110):** | | | | |
|---|---|---|---|---|
| Offset | Length | Type | Value | Description |
| 0 | 1 | const8 | 0x7f | Library opcode |
| 1 | 4 | handle | | Library handle |
| 5 | 4 | long | | Reserved |
| 9 | 1..101 | string | | Reserved |
| **Returns (packet size = 8):** | | | | |
| 0 | 1 | ubyte | | Transaction number |
| 1 | 1 | const8 | 0x7f | Library opcode |
| 2 | 2 | ushort | | Library error code |
| 4 | 4 | handle | | Transaction handle |

### 0E.FF Delete Extension

(no API equivalent)

| **Command:** | |
|---|---|
| data1: | CMD2 value of extension to delete |
| **Immediate Reply:** | |
| Transaction response | |
| **Transaction Reply:** | |
| data1: | transaction number |
| data2: | 0 for success, else error |
| [data 3-6: | extended error code] |
| 0E.xx | Extension Command Execution (.xx is the extension ID) |

### 0E.xx Extension Command Execution

(no API equivalent)

| **Command:** | |
|---|---|
| data1: | PLS serial command ID |
| data2-n: | PLS serial command specific input data |
| **Immediate Reply:** | |
| Transaction response | |
| **Transaction Reply:** | |
| data1: | transaction number |
| data2: | PLS serial command ID |
| data3-4: | PLS serial command error response (ZERO indicates no error) |
| data5-n: | PLS serial command specific result data |
| **Comments:** | |
| The .xx in the command number is the extension ID. | |

# Port access commands (0x10)

Port access commands are used to:

- Open and close ports
- Retrieve port status
- Set play/record actions of attached movies

The commands are specific to a single port, but are not specific to a particular movie or media file.

### 10.00 Get Port Status

(bitmask for calls)

| **Command:** | | | |
|---|---|---|---|
| data1-4: | Handle of port for which status is to be returned | | |
| data5-6: | A mask indicating what status information is to be returned: | | |
| | bit0: | Play/record state | (*VdrGetCurrentState*) |
| | bit1: | Current position (field number) along timeline | (*VdrGetPosition*) |
| | bit2: | Movie at current position | (*VdrGetMovieAtPosition*) |
| | bit3: | Motion-play mode | (*VdrGetPlayMode*) |
| | bit4: | Still-play mode | (*VdrGetStillPlayMode*) |
| | bit5: | Current play rate | (*VdrGetCurrentRate*) |
| | bit6: | Field dominance | (*VdrGetFieldDominance*) |
| | bit7-bit15: | Reserved | |

### 10.00 Get Port Status (Continued)

(bitmask for calls)

**Reply:**

data1-2: A copy of the request mask from command data5-6 (if the bit is not implemented it is cleared)

data3-n: Status values requested (value corresponding to the lowest bit in the mask first):

Play/record state (1 byte). One of the following values:

- 0: Idle
- 1: Play-cued
- 2: Record-cued
- 3: Jog-play
- 4: Jog-record
- 5: Shuttle-play
- 6: Shuttle-record

Current position along timeline (integer, 4 bytes)

Current movie handle on timeline (handle, 4 bytes)

Motion-play mode (1 byte). One of the following values:

- 0: Normal play
- 1: Loop play
- 2: Bounce play
- 3: Limited play
- 4: Slaved play

Still-play mode (1 byte). One of the following values:

- 0: Still play by field
- 1: Still play by frame

Current play rate (float, 4 bytes)

Field dominance

- 0: Neither
- 1: First field dominance
- 2: Second field dominance

**Comments:**

Returns the status of the specified port.

### 10.01 Open Port

(*VdrOpenPortConnection*)

**Command:**

| | |
|---|---|
| data1-8: | Named Configuration Space base file name |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | |
|---|---|
| data1: | Transaction number |
| data2-5: | Handle for open port. This value is 0 if the port cannot be opened. |

If data2-5 is 0, data6-9 are also returned:

| | |
|---|---|
| data6-9: | Error code |

**Comments:**

This command opens a port to control the Profile recorder's video and audio systems. It can be used with or without arguments. The optional argument is a null-terminated character string designating a previously stored named configuration space.

This command must be called first to get a port handle that can be used with other commands.

### 10.02 Close Port

(*VdrClosePort*)

**Command:**

| | |
|---|---|
| data1-4: | Handle of port |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command closes a Profile system's control port. The handle is not valid for use with other commands after this call. This command will free the port so it can be used in subsequent **Open Port** commands.

A **Close Port** call releases all resource handles obtained via the **Allocate Resource** call for the port. Calling **Release Resource** detaches all media handles for the resource. The resource and media handles thus detached cannot be used after these calls.

### 10.03 Idle

(*VdrIdle*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comments:** | | |
| This command stops audio and video recording or playback. | | |

### 10.04 Cue Play

(*VdrCuePlay*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| data5-8: | (Optional) Expected shuttle speed (float), assumed 1.0 if not specified | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comments:** | | |
| This command cues up for playback all video or audio media files that have been attached to channels by Attach Media calls from this port. | | |

### 10.05 Shuttle Play

(*VdrShuttle*, state-sensitive)

**Command:**

| | |
|---|---|
| data1-4: | Handle of port |
| data5-8: | Shuttle speed (float); if bytes 5-8 are not present, the speed 1.0 is assumed |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command causes motion playback. If the Profile system is playing in motion mode, this command changes the playback rate. If **Cue Play** has been issued previously, playback begins at the start of the next external reference vertical interval.

If this command is issued from an idle state, start-of-playback latency is not guaranteed.

### 10.06 Jog Play

(*VdrJog*, state-sensitive)

**Command:**

| | |
|---|---|
| data1-4: | Handle of port |
| data5-8: | Number of fields to jog (integer) |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command causes playback to occur in still mode.

If *Number of fields to jog* is not zero, this command steps the image by the number of fields. If this command is issued from an idle state, start-of-playback latency is not guaranteed.

### 10.07 Cue Record

(*VdrCueRecord*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

**Comments:**

This command cues up for recording all video or audio media files that have been attached to channels by **Attach Media** calls from this port.

### 10.08 Shuttle Record

(*VdrShuttle*, state-sensitive)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| data5-8: | Shuttle speed (float); if bytes 5-8 are not present, the speed 1.0 is assumed | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

**Comments:**

This command causes motion recording. If the Profile system is recording in motion mode, this command changes the record rate. If **Cue Record** has been previously issued, recording begins at the start of the next external reference vertical interval. If this command is issued from an idle state, start-of-record latency is not guaranteed.

### 10.09 Jog Record

(*VdrJog*, state-sensitive)

**Command:**

| | |
|---|---|
| data1-4: | Handle of port |
| data5-8: | Number of fields to jog (integer) |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command causes recording to occur in still mode.

If *Number of fields to jog* is not zero, this command steps the time-line position by the number of fields. If this command is issued from an idle state, start-of-record latency is not guaranteed.

### 10.0A Set Position

(*VdrSetPosition* or *VdrSetPositionRelative*)

**Command:**

| | |
|---|---|
| data1-4: | Handle of port |
| data5-8: | Field number of new position |

The following data bytes are only used for a SetPositionRelative version of the command.

| | |
|---|---|
| data9-12: | Handle of the master port |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command causes the current time-line position to be set as specified.

### 10.0B Set Motion Play/Record Mode

(*VdrSetPlayMode*)

**Command:**

| | | |
|---|---|---|
| data1-4: | Handle of port | |
| data5: | Motion play mode. One of the following: | |
| | 0: | Normal play |
| | 1: | Loop play |
| | 2: | Bounce play |
| | 3: | Limited play |
| | 4: | Slaved play |
| (data6-13: | Transmitted for slave play only) | |
| data6-9: | Handle of master port which determines position | |
| data10-13: | Position of slave port relative to master port (integer) | |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command causes motion play to occur in the given mode. The play mode is volatile, defaulting to normal.

- *Play* starts at the current position. If play is forward, it encounters the larger limit field first. If play is backward, it encounters the smaller limit field first.
- *Normal play* causes play to stop at a media limit.
- *Limited play* causes play to stop at a clock limit.
- *Loop play* causes play to continue from the opposite clock limit repeatedly when it reaches a clock limit.
- *Bounce play* causes play to reverse direction when it reaches a clock limit.

Media limits are defined to be:

- The smallest-numbered field on the timeline referenced by an *in-point* of any attached media.
- The largest-numbered field on the timeline referenced by an *out-point* of any attached media.

Clock limits are defined to be:

- The minimum field number set by the **Set Position Limits** command.
- The field number before the maximum field number set by the **Set Position Limits** command.

The default clock value is 0. Duration of play with neither clock limit set is 0.

### 10.0C Set Still Play Mode

(*VdrSetStillPlayMode*)

**Command:**

| | | |
|---|---|---|
| data1-4: | Handle of port | |
| data5: | Still play mode. One of the following: | |
| | 0: | Still play by field |
| | 1: | Still play by frame |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command sets the playback mode to be used during still play.

When *Play By Field* is selected, playback occurs by line doubling the current field. This reduces motion effects, but at the expense of vertical resolution. When *Play By Frame* is selected, playback occurs by playing the current and next field in sequence. This enhances vertical resolution, but at the possible expense of added motion effects. This function has no effect on record mode.

### 10.0D Set Position Limits

(*VdrSetPositionLimits*)

**Command:**

| | |
|---|---|
| data1-4: | Handle of port |
| data5-8: | Minimum position limit |
| data9-12: | Maximum position limit |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command sets the minimum field number and the field after the last field number that can be reached during playback or record.

### 10.0E Get Position Limits

*(VdrGetPosition)*

| **Command:** | |
|---|---|
| data1-4: | Handle of port |
| **Immediate Reply:** | |
| data1-4: | Minimum position limit |
| data5-8: | Maximum position limit |
| **Comments:** | |

This command returns the minimum field number and the maximum field number (maximum clock limit) that can be reached during playback or record.

### 10.0F Get Next Open Port

*(VdrGetNextPort)*

| **Command:** | |
|---|---|
| data1-4: | Handle of port. The reply contains the handle of the port following this port. |
| **Immediate Reply:** | |
| data1-4: | Next port handle (0 indicates end of enumeration) |
| If data1-4 is 0, data5-8 are also returned: | |
| data5-8: | Error code |
| **Comments:** | |

This command returns the handle of the port following the current port. To get the handle of the first port, use value 0.

### 10.10 Get Previous Open Port

*(VdrGetPreviousPort)*

| **Command:** | |
|---|---|
| data1-4: | Handle of port; the reply contains the handle of the port prior to this port |
| **Immediate Reply:** | |
| data1-4: | Previous port handle (0 indicates end of enumeration) |
| If data1-4 is 0, data5-8 are also returned: | |
| data5-8: | Error code |
| **Comments:** | |

This command gets the handle of the port prior to the current port. To get the handle of the last port, use value 0.

### 10.11 Set Field Dominance

(*VdrSetFieldDominance*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| data5: | Desire dominance of the following types: | |
| | 0: | No dominance |
| | 1: | Field 1 dominance |
| | 2: | Field 2 dominance |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 10.12 Reset Port

(*VdrResetPort*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 10.13 Set Default Tc Format

(*VdrSetDefaultTcFormat*)

*Engineering specification not yet available.*

### 10.14 Shuttle At Gen Tc

(*VdrShuttleAtGenTc*

*Engineering specification not yet available.*

### 10.15 Cancel At Gen Tc Cmds

(*VdrCancelPendingTimecodeTriggerCommands*)

*Engineering specification not yet available.*

# PDR movie access commands (0x21)

PDR movie access commands are used to access movies on the storage subsystem.

### 21.00 Pdr Get Movie Status

(bitmask for calls)

| **Command:** | | | |
|---|---|---|---|
| data1-4: | Movie token | | |
| data5-6: | Mask of desired movie information as follows: | | |
| | bit0: | Movie attributes | (*PdrGetMovieAttributes*) |
| | bit1: | Movie length | (*PdrGetMovieLength*) |
| | bit2: | Movie creation time | (*PdrGetMovieCreateTime*) |
| | bit3: | Movie last change time | (*PdrGetMovieLastChangeTime*) |
| | bit4: | Movie name | (*PdrGetMovieName*) |
| | bit5: | Movie group | (*PdrGetMovieGroup*) |
| | bit6: | Movie dataset | (*PdrGetMovieDataset*) |
| | bit7: | Movie mark-in | (*PdrGetMovieMarkIn*) |
| | bit8: | Movie mark-out | (*PdrGetMovieMarkOut*) |
| data7: | Length type. 0 is minimum, 1 is maximum. (This byte is optional if the Pdr Movie Length is requested.) | | |
| **Reply:** | | | |
| data1-2: | A copy of the request mask from command data5-6 (any request not implemented causes the corresponding mask bit to be cleared) | | |
| data3-n: | Status values requested in the following order (lowest bit first): | | |
| | Movie attributes (4 bytes): | | |
| | 0x00000001 | PdrReadOnly | |
| | 0x00000002 | PdrControlRO | |
| | 0x00000004 | PdrLocked | |
| | 0x00010000 | PdrOpen | |
| | 0x00020000 | PdrOpenMultiple | |
| | 0x00040000 | PdrOpenExclusive | |
| | 0x00080000 | PdrSimpleClip | |
| | 0x10000000 | PdrError | |
| | Movie length (integer, 4 bytes) | | |
| | Movie creation time (date, time 4 bytes total) | | |
| | Movie last modification time (date, time 4 bytes total) | | |
| | Movie name | (string, null terminated) | |
| | Group name | (string, null terminated) | |
| | Dataset name | (string, null terminated) | |
| | Movie mark-in (integer, 4 bytes) | | |
| | Movie mark-out (integer, 4 bytes) | | |

### 21.01 Pdr Find First Dataset

*(PdrFindFirstDataset)*

| **Command:** | |
|---|---|
| (no data bytes) | |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Token for subsequent calls (value < 1 indicates error) |
| If data2-5 is a valid token, data6-n are returned: | |
| data6-n: | The name of the first dataset |
| If data2-5 indicate an error, data6-9 are returned: | |
| data6-9: | Error code |

### 21.02 Pdr Find Next Dataset

*(PdrFindNextDataset)*

| **Command:** | |
|---|---|
| data1-4: | Token from previous call |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-n: | Name of the next dataset (Null, if none) |
| If data2 is Null, data3-6 are also returned: | |
| data3-6: | Error code |

### 21.03 Pdr Close Find

*(PdrCloseFind)*

| **Command:** | | |
|---|---|---|
| data1-4: | Token from Find First Dataset, Group, or Movie call | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 21.04 Pdr Get Current Dataset

*(PdrGetCurrentDataset)*

| **Command:** | |
|---|---|
| data1-n: | Dataset name to get as the current dataset. (This could be Null for all datasets, or just a dataset name for all groups within that dataset, or a combination name of dataset: for that explicit directory.) |
| **Reply:** | |
| data1-n: | The name of the current dataset (Null indicates error) |
| If data1 is Null, data2-5 are also returned: | |
| data2-5: | Error code |

### 21.05 Pdr Set Current Dataset

*(PdrSetCurrentDataset)*

| **Command:** | | |
|---|---|---|
| data1-n: | Dataset name to set as current dataset. (This could be Null for all datasets, or just a dataset name for all groups within that dataset, or a combination name of dataset for that explicit directory.) | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 21.06 Pdr Find First Group

*(PdrFindFirstGroup)*

| **Command:** | |
|---|---|
| data1-n: | Dataset name to walk through |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Token for subsequent calls (value < 1 indicates error) |
| If data2-5 is a valid token, data6-n are returned: | |
| data6-n: | The name of the first group |
| If data2-5 indicate an error, data6-9 are returned: | |
| data6-9: | Error code |

### 21.07 Pdr Find Next Group

*(PdrFindNextGroup)*

| **Command:** | |
|---|---|
| data1-4: | Token from previous call |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-n: | Name of the next group (Null, if none) |
| If data2 is Null, data3-6 are also returned: | |
| data3-6: | Error code |

### 21.08 Pdr Get Current Group

*(PdrGetCurrentGroup)*

| **Command:** | |
|---|---|
| (no data bytes) | |
| **Reply:** | |
| data1-n: | The name of the current group. (Null indicates error.) |
| If data1 is Null, data2-5 are also returned: | |
| data2-5: | Error code |

### 21.09 Pdr Set Current Group

*(PdrSetCurrentGroup)*

| **Command:** | | |
|---|---|---|
| data1-n: | Dataset or group name to set as current group (dataset). (This could be Null for all datasets and groups, or just a dataset name for all groups within that dataset, or a combination name of Dataset:/Group/ for that explicit directory.) | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 21.0A Pdr Find First Movie

(*PdrFindFirstMovie*)

| **Command:** | |
|---|---|
| data1-n: | Dataset/group name to walk through |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Token for subsequent calls (value < 1 indicates error) |
| If data2-5 is a valid token, data6-n are returned: | |
| data6-n: | The name of the first movie |
| If data2-5 indicate an error, data6-9 are returned: | |
| data6-9: | Error code |

### 21.0B Pdr Find Next Movie

(*PdrFindNextMovie*)

| **Command:** | |
|---|---|
| data1-4: | Token from previous call |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-n: | Name of the next movie (Null, if none) |
| If data2 is Null, data3-6 are also returned: | |
| data3-6: | Error code |

### 21.0C Pdr Create Movie

(*PdrCreateMovie*)

| **Command:** | |
|---|---|
| data1-n: | Complex movie name |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | |
| If data2-5 is Null, data6-9 are also returned: | |
| data6-9: | Error code |

### 21.0D Pdr Open Movie

*(PdrOpenMovie)*

| **Command:** | | |
|---|---|---|
| data1-n: | Complex movie name | |
| data(n+1)-(n+4): | PdrExclusiveFlag | 00 00 00 40 |
| | else | 00 00 00 00 |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2-5: | Movie token for existing movie (Null indicates error) | |
| If data2-5 is Null, data6-9 are also returned: | | |
| data6-9: | Error code | |

### 21.0E Pdr Close Movie

*(PdrCloseMovie)*

| **Command:** | | |
|---|---|---|
| data1-4: | Movie token | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 21.0F Pdr Copy Movie Token

*(PdrCopyMovieToken)*

| **Command:** | |
|---|---|
| data1-4: | Existing movie token |
| data5-n: | New complex movie name |
| **Reply:** | |
| data1-4: | Movie token for new movie (Null indicates error) |
| If data1-4 is Null, data5-8 are also returned: | |
| data5-8: | Error code |

### 21.10 Pdr Rename Movie

(*PdrRenameMovie*)

| **Command:** | | |
|---|---|---|
| data1-n: | Complex movie name | |
| data(n+1)-(n+x): | New movie name | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 21.11 Pdr Delete Movie

(*PdrDeleteMovie*)

| **Command:** | | |
|---|---|---|
| data1-n: | Complex movie name | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 21.12 Pdr Get Movie State

(*PdrGetMovieStateInfo*)

| **Command:** | | |
|---|---|---|
| data1-n: | Complex movie name | |
| **Reply:** | | |
| data1: | Status of call | |
| | 0: | OK |
| | 1: | Error condition |
| data2-5: | Movie attributes (see 21.00, *Pdr Get Movie Status* for breakdown) | |
| data6-9: | Minimum length | |
| data10-13: | Maximum length | |
| data14-17: | Movie creation time (date, time 2 bytes each) | |
| data18-21: | Movie last modification time (date, time 2 bytes each) | |
| data22: | Number of JPEG video tracks | |
| data23: | Number of audio tracks | |
| data24: | Number of timecode tracks | |
| data25-28: | Exclusive process ID | |
| data29-32: | Movie mark-in | |
| data33-36: | Movie mark-out | |
| If data1 indicates an error, data3-6 are returned: | | |
| data3-6: | Error code | |

### 21.13 Pdr Set Movie Attributes

(bitmask for calls)

| **Command:** | | | |
|---|---|---|---|
| data1-4: | Movie token | | |
| data5: | Read-write action flag as follows: | | |
| | 0x00 | Set nothing | (no action) |
| | 0x01 | Set movie RW | (*PdrSetMovieReadWrite*) |
| | 0x02 | Set movie RO | (*PdrSetMovieReadOnly*) |
| data6: | Locking action flag as follows: | | |
| | 0x00 | Set nothing | (no action) |
| | 0x01 | Set movie locked | (*PdrSetMovieLocked*) |
| | 0x02 | Set movie unlocked | (*PdrSetMovieUnlocked*) |
| **Reply:** | | | |
| data1: | First flag specified (R/W first the Locking): | | |
| | 0: | OK | |
| | 1: | Error condition | |
| If data1 indicates an error, data2-5 are also returned: | | | |
| data2-5: | Error code | | |

### 21.14 Pdr Get Number of Movies

(*PdrGetNumMovies*)

| **Command:** | | |
|---|---|---|
| data1-n: | Dataset/group name | |
| **Reply:** | | |
| data1-4: | Number of movies in specified area. (Value < 0 indicates error.) | |
| If data1-4 indicates an error, data5-8 are also returned: | | |
| data5-8: | Error code | |

### 21.15 Pdr Movie Exists

(*PdrMovieExists*)

| **Command:** | | |
|---|---|---|
| data1-n: | Complex name of movie | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK (it exists) |
| | 1: | Error condition (it does not exist) |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 21.16 Pdr Set Clip Marks

(bitmask for calls)

| **Command:** | | | |
|---|---|---|---|
| data1-2: | One of the following bitmaps: | | |
| | bit0: | Set mark-in | (*PdrSetClipIn*) |
| | bit1: | Set mark-out | (*PdrSetClipOut*) |
| data3-6: | Movie token | | |
| data7-10: | Mark-in point if specified, else mark-out point | | |
| data11-14: | Mark-out point only if both specified, else not used. | | |
| **Reply:** | | | |
| data1: | Status of call: | | |
| | 0: | OK | |
| | 1: | Error condition | |
| If data1 indicates an error, data2-5 are also returned: | | | |
| data2-5: | Error code | | |

### 21.17 Pdr Trim Clip

(*PdrTrimClip*)

| **Command:** | | |
|---|---|---|
| data1-4: | Movie token | |
| data5-8: | Mark-in point | |
| data9-12: | Mark-out point | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 21.18 Pdr Copy Movie

(*PdrCopyMovie*)

| **Command:** | | |
|---|---|---|
| data1-n: | NULL terminated name of the source machine | |
| data(n+1)-(n+x): | Source complex movie name | |
| data(n+x+1)-(n+z): | Destination complex movie name | |
| data(n+z+1): | Type of copy as follows: | |
| | 0: | (PdrShareMedia) |
| | 1: | (PdrRenderedMedia) |
| | 2: | (PdrExactMedia) |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2-5: | Wait token (0 if error) | |
| If data2-5 indicates an error, data6-9 are also returned: | | |
| data6-9: | Error code | |

### 21.19 Pdr Save Movie

*(PdrSaveMovie)*

| **Command:** | |
|---|---|
| data1-n: | Complex movie name |
| data(n+1): | Number of resources to use |
| data(n+2)-(n+x): | Resource handles to use for attaching (maximum of 32) |
| data(n+x+1)-(n+x+4): | Start field |
| data(n+x+5)-(n+x+8): | End field |
| data(n+x+9): | Copy type (See prior call for breakdown) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Wait token (0 if error) |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |

### 21.1A Pdr Get Wait Operation Status

*(PdrGetWaitOpStatus)*

| **Command:** | | |
|---|---|---|
| data1-4: | Wait token (returned from save or copy movie) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an OK status, data3-23 are returned: | | |
| data3-6: | Percent completed | |
| data7-10: | Average bandwidth | |
| data11-14: | Last bandwidth | |
| data15: | Wait operation state as follows: | |
| | 0: | PdrWaitOpError |
| | 1: | PdrWaitOpRequest |
| | 2: | PdrWaitOpCanceled |
| | 3: | PdrWaitOpContinue |
| | 4: | PdrWaitOpInvalidHandle |
| | 5: | PdrWaitOpCompleted |
| data16-19: | Number of file copying | |
| data20-23: | Number of file waiting | |
| If data2 indicates an error, data3-6 are returned: | | |
| data3-6: | Error code | |

### 21.1B Pdr Terminate Wait Operation

(*PdrTerminateWaitOperation*)

| **Command:** | | |
|---|---|---|
| data1-4: | Wait token (returned from save or copy movie) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an OK status, data3-23 are returned: | | |
| data3-6: | Percent completed | |
| data7-10: | Average bandwidth | |
| data11-14: | Last bandwidth | |
| data15: | Wait operation state as follows: | |
| | 0: | PdrWaitOpError |
| | 1: | PdrWaitOpRequest |
| | 2: | PdrWaitOpCanceled |
| | 3: | PdrWaitOpContinue |
| | 4: | PdrWaitOpInvalidHandle |
| | 5: | PdrWaitOpCompleted |
| data16-19: | Number of file copying | |
| data20-23: | Number of file waiting | |
| If data2 indicates an error, data3-6 are returned: | | |
| data3-6: | Error code | |

### 21.1C Pdr Set User Data

(*PdrSetUserData*)

| **Command:** | |
|---|---|
| data1: | Field buffer number (0 if first time, else value returned with previous call) |
| data2-n: | Actual data to append (max 250 bytes per call) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Buffer number (0 = error) |

### 21.1D Pdr Set User Data Commit

*(PdrSetUserData)*

| **Command:** | |
|---|---|
| data1-4: | Movie token |
| data5-6: | Track token |
| data7-14: | Media token |
| data15-18: | Tag |
| data19: | Field buffer number (Returned from prior call) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |

### 21.1E Pdr Get User Data

*(PdrGetUserData)*

| **Command:** | |
|---|---|
| data1-4: | Movie token |
| data5-6: | Track token |
| data7-14: | Media token |
| data15-18: | Tag |
| data19-22: | Length of user data |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Memory pointer (Null indicates error) |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |

### 21.1F Pdr Get User Data Commit

(*PdrGetUserData*)

| **Command:** | | |
|---|---|---|
| data1-4: | Memory pointer (Returned from prior call plus amount already transferred.) | |
| data5-8: | Amount of data to transfer (250 bytes maximum) | |
| data9: | Completion flag (1 if last read call, else 0) | |
| data10-13: | Starting memory pointer to be released (needed only if completion flag is 1) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Bad starting memory pointer for release |
| data3-n: | Actual data | |

### 21.20 Pdr Backup Movie Descriptor

(*PdrBackupMovieDescription*)

| **Command:** | | |
|---|---|---|
| data1-n: | Complex movie name | |
| data(n+1)-(n+4): | Flags | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 21.21 Pdr Restore Movie Descriptor

(*PdrRestoreMovieDescription*)

| **Command:** | |
|---|---|
| data1-n: | Complex movie name |
| data(n+1)-(n+4): | Flags |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |

### 21.22 Pdr Get List of Names

(*PdrGetListOfNames*)

| **Command:** | |
|---|---|
| data1-n: | Dataset/group name. (See *PdrSetCurrentGroup* on page 179 for name explanation) |
| data(n+1)-(n+4): | Starting index |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Number of names in buffer (250 maximum). -1 indicates error. |
| If data2 is valid, data3-n are returned: | |
| data3-n: | A series of null terminated names |
| If data2 indicates an error, data3-6 are returned: | |
| data3-6: | Error code |

### 21.23 Pdr Get Movie Changes

(*PdrGetMovieChanges*)

| **Command:** | | |
|---|---|---|
| data1: | Event type, one of the following: | |
| | 1: | PdrMovieAllChanges |
| | 2: | PdrMovieAdded |
| | 3: | PdrMovieDeleted |
| | 4: | PdrMovieModified |
| data2: | Number of structures to be returned. | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Number of information structures returned | |
| data3-n: | The actual PdrEventInfo structures with data breakdown: | |
| | Event type: | 1 byte, 0=error |
| | Event time: | 4 bytes in MS-DOS Date/Time format |
| | Modify bitmask: | 4 Bytes |
| | Character String (null terminated) with movie name | |

### 21.24 Pdr Create Event

(*PdrCreateEvent*)

| **Command:** | | |
|---|---|---|
| data1: | Type of event to create, one of the following: | |
| | 1: | PdrMovieAllChanges |
| | 2: | PdrMovieAdded |
| | 3: | PdrMovieDeleted |
| | 4: | PdrMovieModified |
| | 5: | PdrMovieGroupChange |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2-5: | Handle of event created (Null indicates error) | |
| If data2-5 indicates an error, data6-9 are also returned: | | |
| data6-9: | Error code | |

### 21.25 Pdr Set Registry

*(PdrSetRegistry)*

| **Command:** | | |
|---|---|---|
| data1-4: | Maximum movies | |
| data5-8: | Maximum media references | |
| data9-12: | Minimum media defines | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 21.26 Pdr Get Registry

*(PdrGetRegistry)*

| **Command:** | | |
|---|---|---|
| (no command data) | | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an OK status, data3-14 are returned: | | |
| data3-6: | Maximum number of movies | |
| data7-10: | Maximum number of media references | |
| data11-14: | Minimum number of media defines | |
| If data2 indicates an error, data3-6 are returned: | | |
| data3-6: | Error code | |

### 21.27 Get First Timecode Position

(*PdrGetFirstTcPosition*)

| **Command:** | |
|---|---|
| data1-n: | Movie name (Null terminated) |
| data(n+1)-(n+4): | Logical track number |
| data(n+5)-(n+8): | Timecode value |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Field number in the movie that matches the first occurrence of the timecode (negative value indicates an error) |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |

### 21.28 Get Last Timecode Position

(*PdrGetLastTcPosition*)

| **Command:** | |
|---|---|
| data1-n: | Movie name (Null terminated) |
| data(n+1)-(n+4): | Logical track number |
| data(n+5)-(n+8): | Timecode value |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Field number in the movie that matches the last occurrence of the timecode (negative value indicates an error) |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |

### 21.29 Rename Open Movie

(*PdrRenameOpenMovie*)

| **Command:** | | |
|---|---|---|
| data1-4: | Movie token | |
| data5-n: | Complex movie name | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 21.2A (Reserved)

### 21.2B Movie To Media Name

(no API equivalent)

**Command:**

| | |
|---|---|
| data1-n: | Complex movie name |
| data(n+1): | Pdr track type (See 40.15 *Get Track Token Type*) |
| data(n+2): | Zero-based track number to specify which track of the given type to use. |
| data(n+3): | Flag to specify how to interpret the next parameter: |
| 0: | The value in data(n+4)-(n+7) is to be used as an ordinal number (starting with 1) of the media file within the specified track of the movie. |
| 1: | The value in data(n+4)-(n+7) is to be used as a timeline position. |
| data(n+4)-(n+7): | Order or timeline position. See comments. |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | |
|---|---|
| data1: | Transaction number |
| data2-n: | Media file pathname, or null. |

If data2 is null, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command returns the media file name of a specific piece of media that is on a specific track of the given movie. The track is specified by type and number. The last two parameters of the command identify the specific media segment from the track.

If the flag in data(n+3) is 0, then the last parameter specifies the relative order of the media. The order parameter is a one-based value; specifying 1 will return the first media on the track, specifying 2 will return the second media segment from the track, etc.

If the flag in data(n+3) is 1, then the last parameter specifies a timeline position. The media file that would be at the given timeline position if the track were attached starting at timeline position 0 is the media that will be identified.

### 21.2C Pdr Set Small User Data

(no API equivalent)

| Command: | | |
|---|---|---|
| data1-4: | Movie token | |
| data5-6: | Track token | |
| data7-14: | Media token | |
| data15-18: | Tag | |
| data19-255: | User data to set. (237 maximum length.) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

### 21.2D Pdr Get Small User Data

(no API equivalent)

| Command: | | |
|---|---|---|
| data1-4: | Movie token | |
| data5-6: | Track token | |
| data7-14: | Media token | |
| data15-18: | Tag | |
| data19-22: | length of data | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| data3-x: | Actual data returned. (None if in error.) | |

### 21.2E Set Open Movie Marks

(bitmask for calls)

| **Command:** | | | |
|---|---|---|---|
| data1-2: | One of the following bitmaps: | | |
| | bit0: | Set mark-in | (*PdrSetMovieMarkIn*) |
| | bit1: | Set mark-out | (*PdrSetMovieMarkOut*) |
| data3-6: | Movie token | | |
| data7-10: | Mark-in point if specified; otherwise, mark-out point. | | |
| data11-14: | Mark-out point only if both specified; otherwise, not used. | | |
| **Reply:** | | | |
| data1: | One of the following: | | |
| | 0: | OK | |
| | 1-255: | Error condition | |

### 21.2F Set Movie Marks

(*PdrSetMovieMarks*)

| **Command:** | | | |
|---|---|---|---|
| data1: | One of the following bitmaps: | | |
| | bit0: | Set mark-in | (PdrMarkIn) |
| | bit1: | Set mark-out | (PdrMarkOut) |
| | Both set | | (PdrMarkBoth) |
| data2-n: | Movie name. (Null terminated.) | | |
| data(n+1)-(n+5): | Mark-in point if specified; otherwise, mark-out point. | | |
| data(n+6)-(n+9): | Mark-out point only if both specified; otherwise, not used. | | |
| **Reply:** | | | |
| data1: | One of the following: | | |
| | 0: | OK | |
| | 1-255: | Error condition | |

### 21.30 Pdr Save Movie By Port

(*PdrSaveMovie*)

| **Command:** | |
|---|---|
| data1-n: | Complex movie name |
| data(n+1)-(n+5): | Port handle |
| data(n+6)-(n+9): | Start field |
| data(n+10)-(n+13): | End field |
| data14: | Copy type |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Wait token (0 if error) |

### 21.31 Pdr Close All Enumerators

(*PdrCloseAllEnumTokens*)

| **Command:** | | |
|---|---|---|
| data1-n: | Complex movie name | |
| data(n+1)-(n+5): | Port handle | |
| data(n+6)-(n+9): | Start field | |
| data(n+10)-(n+13): | End field | |
| data14: | Copy type | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Status of call: | |
| | 0 | OK |
| | 1 | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 21.32 Pdr Get Movie State Info

*(PdrGetMovieStateInfo)*

| **Command:** | |
|---|---|
| data1-n: | Complex movie name |
| **Reply:** | |
| data1: | Status of call (0 = OK, else error) |
| data2-5: | Movie attributes (see 21.00, *Pdr Get Movie Status* for breakdown) |
| data6-9: | Minimum length |
| data10-13: | Maximum length |
| data14-17: | Movie creation time (date, time 2 bytes each) |
| data18-21: | Movie last modification time (date, time 2 bytes each) |
| data22: | Number of JPEG video tracks |
| data23: | Number of audio tracks |
| data24: | Number of timecode tracks |
| data25-28: | Exclusive process ID |
| data29-32: | Movie mark-in |
| data33-36: | Movie mark-out |
| data37: | Number of MPEG video tracks |
| data38-41: | First timecode value found on first timecode track |
| data42-45: | Last timecode value found on first timecode track |
| data46: | Mpeg chroma: |
| | 0: MpegChroma420 |
| | 1: MpegChroma422 |
| | 2: MpegChromaUnknown |
| data47: | Mpeg GOP end: |
| | 0: GopOpenEnd |
| | 1: GopClosedEnd |
| | 2: GopEndUnknown |
| data48: | Mpeg pix structure: |
| | 0: PixStructureFrame |
| | 1: PixStructureField |
| | 2: PixTypeUnknown |
| data49: | MpegIPixPerGop |
| data50: | MpegPPixPerIPix |
| data51: | MpegBPixPerIPPix |

# Attached movie access commands (0x30)

Attached movie access commands are used to:

- Place and remove movies from the timeline
- Retrieve current attached movie status

### 30.00 Get Attached Movie Status

(bitmask for calls)

**Command:**

| | | | |
|---|---|---|---|
| data1-4: | Handle of movie for which status is needed | | |
| data5-6: | Mask indicating what status information is to be returned: | | |
| | bit0: | Starting position of movie (field number on timeline) | (*VdrGetMovieStartPosition*) |
| | bit1: | Ending position of movie (field number on timeline) | (*VdrGetMovieEndPosition*) |
| | bit2: | Mark-in point (record number in media files) | (*VdrGetMovieMarkIn*) |
| | bit3: | Mark-out point (record number in media files) | (*VdrGetMovieMarkOut*) |
| | bit4: | Movie name | (*VdrGetMovieName*) |
| | bit5: | Mark duration (mark-out point minus mark-in point) | (*VdrGetMovieMarkDuration*) |
| | bit6: | Movie user data | (*VdrGetMovieUserData*) |
| | bit7: | Movie access mode | (*VdrGetMovieAccessMode*) |
| | bit8: | Movie default mark-in | (*VdrGetMovieDefaultMarkIn*) |
| | bit9: | Movie default mark-out | (*VdrGetMovieDefaultMarkOut* ) |
| | bit10-15: | Reserved | |

**Immediate Reply:**

Ack + transaction number

### 30.00 Get Attached Movie Status (Continued)

(bitmask for calls)

**Transaction Reply:**

| | |
|---|---|
| data1: | Transaction number |
| data2-3: | A copy of the request mask from command data5-6 (any request not implemented causes the corresponding bit in the mask to be cleared) |
| data4-n: | Status values requested (value corresponding to the lowest bit in the mask first): |
| | Starting position for movie (integer, 4 bytes) |
| | Ending position for movie (integer, 4 bytes) |
| | Mark-in point for movie (integer, 4 bytes) |
| | Mark-out point for movie (integer, 4 bytes) |
| | Name of movie (null-terminated string) |
| | Mark duration for movie (integer, 4 bytes) |
| | User data (integer, 4 bytes) |
| | Access mode (1 byte) as follows: |
| | 0: Access read only |
| | 1: Access read write |
| | Default mark-in point for movie (integer, 4 bytes) |
| | Default mark-out point for movie (integer, 4 bytes) |

**Comments:**

This command gets the status of the attached movie.

Starting position for movie returns the reference clock position where the movie handle begins to play or record. Ending position for movie returns the reference clock position at the field following the last field played or recorded in the movie handle. Mark-in point returns the in point for the movie handle (the first field in the media files to be played or recorded for the movie). Mark-out point returns the out point for the movie handle (the field after the last field in the media files to be played or recorded for the movie).

### 30.01 Attach Movie

*(VdrAttachMovie)*

| **Command:** | |
|---|---|
| data1-n: | Name of movie (null terminated) |
| data(n+1)-(n+4): | Handle of port to which movie is to be attached |
| **Optional data items:** | |
| data(n+4)-(n+8): | Handle of movie before which this movie should be attached. Use value 0 to attach at end of movie list. |
| data(n+9): | Shift mode. One of the following:<br>0: Any movies that were attached before the data(n+4)-(n+8) handle will be shifted to an earlier timeline position.<br>1: Any movies that were attached after and/or include the data(n+4)-(n+8) handle will be shifted to a later timeline position.<br>If the handle is Null, ShiftAfter has no effect on other movies, while ShiftBefore effects *all* other movies by moving them to an earlier position on the timeline. |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Handle of attached movie. (0 if movie could not be attached.) |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |
| **Comments:** | |
| This command attaches the movie (a collection of media files) to the resources of a port for playback or record. If there are no arguments after the port handle, the movie is attached at the end of the list. | |

### 30.02 Attach Movie with Marks

*(VdrAttachMovieWithMarks)*

| **Command:** | |
|---|---|
| data1-n: | Name of movie (null terminated) |
| data(n+1)-(n+4): | Mark-in (integer) |
| data(n+5)-(n+8): | Mark-out (integer) |
| data(n+9)-(n+13): | Handle of port to which movie is to be attached |
| **Optional data items:** | |
| data(n+14)-(n+17): | Handle of movie before which this movie should be attached. Use value 0 to attach at end of list of movies |
| data(n+18): | Shift mode. One of the following: |
| | 0: Any movies that were attached before the data(n+14)-(n+17) handle will be shifted to an earlier timeline position. |
| | 1: Any movies that were attached after and/or include the data(n+14)-(n+17) handle will be shifted to a later timeline position. |
| | If the handle is Null, ShiftAfter has no effect on other movies, while ShiftBefore effects *all* other movies by moving them to an earlier position on the timeline. |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Handle of attached movie. (0 if movie could not be attached.) |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |
| **Comments:** | |
| This command attaches the movie (a collection of media files) to the resources of the port for record and playback, and sets the mark-in and mark-out point for the movie. If there are no arguments after the port handle, the movie is attached at the end of the list. | |

### 30.03 Detach Movie

(*VdrDetachMovie*)

**Command:**

| | | |
|---|---|---|
| data1-4: | Handle of movie to detach | |
| data5: | Shift mode. One of the following: | |
| | 0: | Any movies that were attached before the data1-4 handle will be shifted to a later timeline position. |
| | 1: | Any movies that were attached after the data1-4 handle will be shifted to an earlier timeline position. |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

The associated files are removed from the playback or record lists for the resources to which they were attached. The files continue to exist in the Profile file system. The movie handle cannot be used after this call. Detaching a movie automatically resets the minimum and maximum play and record positions.

### 30.04 Get Next Movie

(*VdrGetNextMovie*)

**Command:**

| | |
|---|---|
| data1-4: | Handle of movie, use 0 to retrieve the first |
| data5-8: | Resource handle. This field is optional; it is necessary only if the handle is 0 and more than one port is open |

**Reply:**

| | |
|---|---|
| data1-4: | Handle of the next movie on the timeline. (0 if the reference movie is the last movie on the timeline.) |

**Comments:**

This command returns the movie handle of a movie that has fields located at the reference position on the timeline. It returns the handle of the movie that follows the reference movie on the timeline.

The reply contains the handle of the movie following this reference movie on the timeline. To get the handle of the first movie on the timeline, use value 0 as the reference movie.

### 30.05 Get Previous Movie

(*VdrGetPreviousMovie*)

| **Command:** | |
|---|---|
| data1-4: | Handle of reference movie, use 0 to retrieve the last |
| data5-8: | Resource handle. This field is optional; it is necessary only if the handle is 0 and more than one port is open |
| **Reply:** | |
| data1-4: | Handle of the previous movie on the timeline. (0 if the reference movie is the first movie on the timeline.) |

**Comments:**

This command returns the handle of the movie that precedes the reference movie on the timeline.

The reply contains the handle of the movie preceding this reference movie on the timeline. To get the handle of the last movie on the timeline, use value 0.

### 30.06 Get Movie at Position

(*VdrGetMovieAtPosition*)

| **Command:** | |
|---|---|
| data1-4: | Handle of the port to which movie is attached |
| data5-8: | Position (field number) on the timeline |
| **Reply:** | |
| data1-4: | Handle of the movie at position on the timeline. (0 indicates an error.) |
| If data1-4 indicates an error, data5-8 are also returned: | |
| data5-8: | Error code |

**Comments:**

The reply contains the handle of a movie that has fields located at this position.

### 30.07 Set Movie Mark In

(*VdrSetMovieMarkIn*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of movie for which mark-in is to be set | |
| data5-8: | Mark-in point (field within movie) | |
| **Optional data item:** | | |
| data9: | Shift mode. One of the following: | |
| | 0: | Any movies that were set for mark-in before the data5-8 handle will be shifted to an earlier timeline position. |
| | 1: | Any movies that were set for mark-in after and/or including the data5-8 handle will be shifted to a later timeline position. |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comments:** | | |

This command sets the mark-in point for the movie referenced by the movie handle for playback or recording.

If there are no arguments after the mark-in point, the mark will be changed with a shift appropriate to the current position.

### 30.08 Set Movie Mark Out

(*VdrSetMovieMarkOut*)

**Command:**

| | |
|---|---|
| data1-4: | Handle of movie for which mark-out is to be set |
| data5-8: | Mark-out point (field within movie) |

**Optional data item:**

| | | |
|---|---|---|
| data9: | Shift mode. One of the following: | |
| | 0: | Any movies that were set for mark-out before the data5-8 handle will be shifted to an earlier timeline position. |
| | 1: | Any movies that were set for mark-out after and/or including the data5-8 handle will be shifted to a later timeline position. |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command sets the mark-out point for the attached movie referenced by the movie handle for playback or recording.

If there are no arguments after the mark-in point, the mark will be changed with a shift appropriate to the current position.

### 30.09 Get Port Handle from Movie Handle

(*VdrGetPortFromMovie*)

**Command:**

| | |
|---|---|
| data1-4: | Handle of movie |

**Reply:**

| | |
|---|---|
| data1-4: | Handle of movie. (Null indicates error.) |

If data1-4 indicates an error, data5-8 are also returned:

| | |
|---|---|
| data5-8: | Error code |

### 30.0A Set Movie User Data

(*VdrSetMovieUserData*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of movie for which the user data is to be set | |
| data5-8: | User data (any 4 bytes) | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 30.0B Attach Movie With Resources

(*VdrAttachMovie*)

| **Command:** | | |
|---|---|---|
| data1-n: | Name of movie (Null terminated) | |
| data(n+1): | Number of resources to use | |
| data(n+2)-(n+x): | Resource handles to use for attaching (maximum of 32) | |
| data(n+x+1)-(n+x+4): | Handle of movie before which this movie should be attached. Use value 0 to attach at end of list of movies | |
| data(n+x+5): | Shift mode. One of the following: | |
| | 0: | Any movies that were attached before the data(n+2)-(n+x) handle will be shifted to an earlier timeline position. |
| | 1: | Any movies that were attached after and/or include the data(n+2)-(n+x) handle will be shifted to a later timeline position. |
| | If the handle is Null, ShiftAfter has no effect on other movies, while ShiftBefore effects *all* other movies by moving them to an earlier position on the timeline. | |
| data(n+x+6): | Mark mode, one of the following: | |
| | 0: | shortest |
| | 1: | longest |
| | 2: | as is |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Handle of attached movie; null indicates error | |
| If data2-5 indicates an error, data6-9 are also returned: | | |
| data6-9: | Error code | |
| **Comments:** | | |
| If there are no arguments after the port handle, the movie will be attached at the end of the list with a shift appropriate to the current position. | | |

### 30.0C Expand Movie

(*VdrExpandMovie*)

**Command:**

| | | |
|---|---|---|
| data1-4: | Handle of movie | |
| data5: | Number of resources to use | |
| data6-n: | Resource handles to use for attaching (maximum of 32) | |
| data(n+1)-(n+4): | Handle of movie before which this movie should be attached. Use value 0 to attach at end of list of movies | |
| data(n+5): | Shift mode. One of the following: | |
| | 0: | Any movies that were attached before the data(n+1)-(n+4) handle will be shifted to an earlier timeline position. |
| | 1: | Any movies that were attached after and/or include the data(n+1)-(n+4) handle will be shifted to a later timeline position. |
| | If the handle is Null, ShiftAfter has no effect on other movies, while ShiftBefore affects all other movies by moving them to an earlier position on the timeline. | |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

If there are no arguments after the port handle, the movie will be attached at the end of the list with a shift appropriate to the current position.

### 30.0D Get Movie at Resource Position

(*VdrGetMovieAtPosition*)

**Command:**

| | |
|---|---|
| data1-4: | Handle of resource to use |
| data5-8: | Field number |

**Reply:**

| | |
|---|---|
| data1-4: | Handle attached movie. (0 if no movie at that position.) |

If data1-4 is 0, data5-8 are also returned:

| | |
|---|---|
| data5-8: | Error code |

### 30.0E Attach Open Movie

(*VdrAttachOpenMovie*)

| **Command:** | | |
|---|---|---|
| data1-4: | Movie token (from PdrOpenMovie) | |
| data5: | Number of resources to use | |
| data6-n: | Resource handles to use for attaching (32 is the maximum) | |
| (If there are no arguments after the resource handles, the movie should be attached at the end of the list with a shift appropriate to the current position) | | |
| data(n+1)-(n+4): | Handle of movie before which this movie should be attached. Use value 0 to attach at end of list of movies | |
| datan+5: | Shift mode. One of the following: | |
| | 0: | Any movies that were attached before the data(n+1)-(n+4) handle will be shifted to an earlier timeline position. |
| | 1: | Any movies that were attached after and/or include the data(n+1)-(n+4) handle will be shifted to a later timeline position. |
| | If the handle is Null, ShiftAfter has no effect on other movies, while ShiftBefore affects *all* other movies by moving them to an earlier position on the timeline. | |
| data(n+6): | Mark mode, one of the following: | |
| | 0: | shortest |
| | 1: | longest |
| | 2: | as is |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2-5: | Handle of attached movie. (0 if movie could not be attached.) | |
| | 0: | OK |
| | 1: | Error condition |
| If data2-5 indicates an error, data6-9 are also returned: | | |
| data6-9: | Error code | |

### 30.0F Disable Movie Position Limits

(*VdrDisableMoviePositionLimiting*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 30.10 Enable Movie Position Limits

(*VdrEnableMoviePositionLimiting*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 30.11 Is Movie Position Limiting Enabled

(*VdrIsMoviePositionLimitingEnabled*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK (it is enabled) |
| | 1: | Error condition (it is not enabled) |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 30.12 Attach Movie With Marks

*(VdrAttachMovieWithMarks)*

| **Command:** | | |
|---|---|---|
| data1-n: | Name of movie (Null terminated) | |
| data(n+1): | Number of resources to use | |
| data(n+2)-(n+x): | Resource handles to use for attaching (maximum of 32) | |
| data(n+x+1)-(n+x+4): | Handle of movie before which this movie should be attached. Use value 0 to attach at end of list of movies | |
| data(n+x+5): | Shift mode. One of the following: | |
| | 0: | Any movies that were attached before the data(n+x+1)-(n+x+4) handle will be shifted to an earlier timeline position. |
| | 1: | Any movies that were attached after and/or include the data(n+x+1)-(n+x+4) handle will be shifted to a later timeline position. |
| | If the handle is Null, ShiftAfter has no effect on other movies, while ShiftBefore effects *all* other movies by moving them to an earlier position on the timeline. | |
| data(n+x+6): | Mark mode, one of the following: | |
| | 0: | shortest |
| | 1: | longest |
| | 2: | as is |
| data(n+x+7)-(n+x+10): | Mark-in | |
| data(n+x+11)-(n+x+14): | Mark-out | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2-5: | Handle of attached movie. (0 if movie could not be attached.) | |
| If data2-5 indicates an error, data6-9 are also returned: | | |
| data6-9: | Error code | |

### 30.13 Attach Open Movie With Marks

*(VdrAttachOpenMovieWithMarks)*

| **Command:** | | |
|---|---|---|
| data1-4: | Movie token (from PdrOpenMovie) | |
| data5: | Number of resources to use | |
| data6-n: | Resource handles to use for attaching (32 is the maximum) | |
| (If there are no arguments after the resource handles, the movie should be attached at the end of the list with a shift appropriate to the current position) | | |
| data(n+1)-(n+4): | Handle of movie before which this movie should be attached. Use value 0 to attach at end of list of movies | |
| datan+5: | Shift mode. One of the following: | |
| | 0: | Any movies that were attached before the data(n+1)-(n+4) handle will be shifted to an earlier timeline position. |
| | 1: | Any movies that were attached after and/or include the data(n+1)-(n+4) handle will be shifted to a later timeline position. |
| datan+6: | Mark mode, one of the following: | |
| | 0: | shortest |
| | 1: | longest |
| | 2: | as is |
| data(n+x+7)-(n+x+10): | Mark-in | |
| data(n+x+11)-(n+x+14): | Mark-out | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2-5: | Handle of attached movie. (0 if movie could not be attached.) | |
| | 0: | OK |
| | 1: | Error condition |
| If data2-5 indicates an error, data6-9 are also returned: | | |
| data6-9: | Error code | |

### 30.14 Expand Movie By Port

(*VdrExpandMovie*)

**Command:**

| | | |
|---|---|---|
| data1-4: | Handle of movie | |
| data5-8: | Number of resources to use | |
| data9-n: | Resource handles to use for attaching (maximum of 32) | |
| data(n+1)-(n+4): | Handle of movie before which this movie should be attached. Use value 0 to attach at end of list of movies | |
| data(n+5): | Shift mode. One of the following: | |
| | 0: | Any movies that were attached before the data(n+1)-(n+4) handle will be shifted to an earlier timeline position. |
| | 1: | Any movies that were attached after and/or include the data(n+1)-(n+4) handle will be shifted to a later timeline position. |
| | If the handle is Null, ShiftAfter has no effect on other movies, while ShiftBefore effects *all* other movies by moving them to an earlier position on the timeline. | |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

If there are no arguments after the port handle, the movie will be attached at the end of the list with a shift appropriate to the current position.

### 30.15 Attach Open Movie By Port

(*VdrAttachOpenMovie*)

**Command:**

| | |
|---|---|
| data1-4: | Movie token (from PdrOpenMovie) |
| data5-8: | Number of resources to use |
| data9-n: | Resource handles to use for attaching (32 is the maximum) |

(If there are no arguments after the resource handles, the movie should be attached at the end of the list with a shift appropriate to the current position)

| | | |
|---|---|---|
| data(n+1)-(n+4): | Handle of movie before which this movie should be attached. Use value 0 to attach at end of list of movies | |
| datan+5: | Shift mode. One of the following: | |
| | 0: | Any movies that were attached before the data(n+1)-(n+4) handle will be shifted to an earlier timeline position. |
| | 1: | Any movies that were attached after and/or include the data(n+1)-(n+4) handle will be shifted to a later timeline position. |
| | If the handle is Null, ShiftAfter has no effect on other movies, while ShiftBefore effects *all* other movies by moving them to an earlier position on the timeline. | |
| datan+6: | Mark mode, one of the following: | |
| | 0: | shortest |
| | 1: | longest |
| | 2: | as is |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | |
|---|---|
| data1: | Transaction number |
| data2-5: | Handle of attached movie. (0 if movie could not be attached.) |

If data2-5 indicates an error, data6-9 are also returned:

| | |
|---|---|
| data6-9: | Error code |

### 30.16 Attach Open Movie With Marks By Port

*(VdrAttachOpenMovieWithMarks)*

| **Command:** | | |
|---|---|---|
| data1-4: | Movie token (from PdrOpenMovie) | |
| data5-8: | Number of resources to use | |
| data9-n: | Resource handles to use for attaching (32 is the maximum) | |
| (If there are no arguments after the resource handles, the movie should be attached at the end of the list with a shift appropriate to the current position) | | |
| data(n+1)-(n+4): | Handle of movie before which this movie should be attached. Use value 0 to attach at end of list of movies | |
| datan+5: | Shift mode. One of the following: | |
| | 0: | Any movies that were attached before the data(n+1)-(n+4) handle will be shifted to an earlier timeline position. |
| | 1: | Any movies that were attached after and/or include the data(n+1)-(n+4) handle will be shifted to a later timeline position. |
| datan+6: | Mark mode, one of the following: | |
| | 0: | shortest |
| | 1: | longest |
| | 2: | as is |
| data(n+x+7)-(n+x+10): | Mark-in | |
| data(n+x+11)-(n+x+14): | Mark-out | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2-5: | Handle of attached movie. (0 if movie could not be attached.) | |
| If data2-5 indicates an error, data6-9 are also returned: | | |
| data6-9: | Error code | |

# Stored media access commands (0x40)

Stored media access commands are used to reference media files stored on the subsystem.

### 40.00 Get Stored Media Status

(bitmask for calls)

| **Command:** | | | |
|---|---|---|---|
| data1-2: | Bitmask of information to be returned: | | |
| | bit0: | Pdr media attributes | (*PdrGetMediaAttributes*) |
| | bit1: | Pdr media data type | (*PdrGetMediaDataType*) |
| | bit2: | Pdr media in point | (*PdrGetMediaIn*) |
| | bit3: | Pdr media out point | (*PdrGetMediaOut*) |
| | bit4: | Pdr media path | (*PdrGetMediaPath*) |
| | bit5: | Pdr media duration | (*PdrGetMediaDuration*) |
| | bit6: | Pdr media token type | (*PdrGetMediaTokenType*) |
| | bit7: | Position of media token | (*PdrGetPositionOfMediaToken*) |
| | bit8: | Movie token from media token | (*PdrGetMovieTokenFromMediaToken*) |
| | bit9: | Get media dataset | (*PdrGetMediaDataset*) |
| | bit10-15: | Reserved | |
| data3-10: | Pdr media token | | |

### 40.00 Get Stored Media Status (Continued)

(bitmask for calls)

| **Reply:** | |
|---|---|
| data1-2: | Copy of bitmask (invalid values are cleared) |
| data3-n: | Status values requested (value corresponding to the lowest bit in the mask first: |
| | Pdr media attributes (4 bytes): |
| | bit0: Read only |
| | bit1: Control read only |
| | bit2: Locked |
| | bit3: Unused |
| | bit4: Non-zero open count |
| | bit5: Open count gone to one |
| | bit6: Exclusive |
| | bit7: Unused |
| | bit8: Error |
| | Pdr media data type (1 byte): |
| | 0: Non-media file |
| | 1: Video |
| | 2: Audio |
| | 3: Timecode |
| | Pdr media in point (4 bytes) |
| | Pdr media out point (4 bytes) |
| | Pdr media path (n bytes with terminating Null) |
| | Pdr media duration (4 bytes) |
| | Pdr media token type (1 byte): |
| | 0: non-media file (error) |
| | 1: track record |
| | 2: media file |
| | 3: media black |
| | Pdr position of media token (4 bytes - field number) |
| | Movie token of media token (4 bytes - movie token) |
| | Media dataset (n bytes with terminating Null) |

### 40.01 Get Maximum Number of Tracks

(*PdrGetMaxTracks*)

| **Command:** | |
|---|---|
| data1: | Track type (Refer to 40.13 for track types.) |
| **Reply:** | |
| data1-4: | Maximum number of tracks of type. (-1 indicates an error.) |
| If data1-4 indicates an error, data5-8 are also returned: | |
| data5-8: | Error code |

### 40.02 Get Number of Tracks

(*PdrGetNumTracks*)

| **Command:** | |
|---|---|
| data1: | Track type |
| data2-5: | Movie token |
| **Reply:** | |
| data1-4: | Number of tracks of type for specified movie. (-1 indicates an error.) |
| If data1-4 indicates an error, data5-8 are also returned: | |
| data5-8: | Error code |

### 40.03 Get Next Track

(*PdrGetNextTrack*)

| **Command:** | |
|---|---|
| data1-4: | Movie token |
| data5-6: | Track token |
| **Reply:** | |
| data1-2: | Next track token. (0 indicates an error.) |
| If data1-2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |

### 40.04 Get Previous Track

(*PdrGetPreviousTrack*)

| **Command:** | |
|---|---|
| data1-4: | Movie token |
| data5-6: | Track token |
| **Reply:** | |
| data1-2: | Previous track token. (0 indicates an error.) |
| If data1-2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |

### 40.05 Get Track Length

(*PdrGetTrackLength*)

| **Command:** | |
|---|---|
| data1-4: | Movie token |
| data5-6: | Track token |
| **Reply:** | |
| data1-4: | Length of track. (-1 indicates an error.) |
| If data1-4 indicates an error, data5-8 are also returned: | |
| data5-8: | Error code |

### 40.06 Get Track Token From Media Token

(*PdrGetTrackTokenFromMediaToken*)

| **Command:** | |
|---|---|
| data1-8: | Media token |
| **Reply:** | |
| data1-2: | Track token. (0 indicates an error.) |
| If data1-2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |

### 40.07 Get Number of Media on Track

(*PdrGetNumMediaOnTrack*)

| **Command:** | |
|---|---|
| data1-4: | Movie token |
| data5-6: | Track token |
| **Reply:** | |
| data1-4: | Number of media on track. (-1 indicates an error.) |
| If data1-4 indicates an error, data5-8 are also returned: | |
| data5-8: | Error code |

### 40.08 Create Media Token

(*PdrCreateMediaToken*)

| **Command:** | |
|---|---|
| data1-4: | Movie token |
| data5-6: | Track token |
| data7-10: | Length |
| data11-18: | Before media token |
| **Reply:** | |
| data1-8: | Next media token. (Null media token is error.) |

### 40.09 Create Black Media Token

(*PdrCreateBlackMediaToken*)

| **Command:** | |
|---|---|
| data1-4: | Movie token |
| data5-6: | Track token |
| data7-10: | Length |
| data11-18: | Before media token |
| **Reply:** | |
| data1-8: | Next media token. (Null media token is error.) |

### 40.0A Create Media Token with Path

(*PdrCreateMediaTokenWithPath*)

| **Command:** | |
|---|---|
| data1-4: | Movie token |
| data5-6: | Track token |
| data7-10: | Length |
| data11-18: | Before media token |
| data19-n: | Path of new media (with null terminator) |
| **Reply:** | |
| data1-8: | Next media token. (Null media token is error.) |

### 40.0B Delete Media

(*PdrDeleteMedia*)

| **Command:** | | |
|---|---|---|
| data1-8: | Media token for media to be deleted | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 40.0C Copy Media Token

(*PdrCopyMediaToken*)

| **Command:** | |
|---|---|
| data1-8: | Media token |
| data9-12: | Movie token |
| data13-14: | Track token |
| data15-18: | Before media token |
| **Reply:** | |
| data1-8: | Next media token. (Null media token is error.) |

### 40.0D Get Next Media Token

(*PdrGetNextMediaToken*)

| **Command:** | |
|---|---|
| data1-4: | Movie token |
| data5-6: | Track token |
| data7-14: | Media token |
| **Reply:** | |
| data1-8: | Next media token. (Null media token is error.) |

### 40.0E Get Previous Media Token

*(PdrGetPreviousMediaToken)*

| **Command:** | |
|---|---|
| data1-4: | Movie token |
| data5-6: | Track token |
| data7-14: | Media token |
| **Reply:** | |
| data1-8: | Previous media token. (Null media token is error.) |

### 40.0F Get Media Token at Position

*(PdrGetMediaTokenAtPosition)*

| **Command:** | |
|---|---|
| data1-4: | Movie token |
| data5-6: | Track token |
| data7-10: | Field number |
| **Reply:** | |
| data1-8: | Media token of media at specified field number. (Null media token is error.) |

### 40.10 Set Media Mark In

*(PdrSetMediaIn)*

| **Command:** | | |
|---|---|---|
| data1-8: | Media token | |
| data9-12: | Field number to use as in-point | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 40.11 Set Media Mark Out

*(PdrSetMediaOut)*

| **Command:** | | |
|---|---|---|
| data1-8: | Media token | |
| data9-12: | Field number to use as out-point | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 40.12 Set Media Marks

(*PdrSetMediaMarks*)

| **Command:** | | |
|---|---|---|
| data1-8: | Media token | |
| data9-12: | Field number to use as in-point | |
| data13-16: | Field number to use as out-point | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 40.13 Create Track Token

(*PdrCreateTrackToken*)

| **Command:** | | |
|---|---|---|
| data1: | Track type, which is one of the following: | |
| | 1: | PdrVideoTrack |
| | 2: | PdrAudioTrack |
| | 3: | PdrTimeCodeTrack |
| | 4: | PdrMpegVideoTrack |
| | 5: | PdrDvcpro25Track |
| | 6: | PdrDvcpro50Track |
| data2-5: | Logical number | |
| **Reply:** | | |
| data1-2: | New track token. (0 = error.) | |

### 40.14 Get Track Token Number

(*PdrGetTrackTokenNum*)

| **Command:** | |
|---|---|
| data1-2: | Track token |
| **Reply:** | |
| data1-4: | Logical unit number of the track. (-1 indicates an error.) |
| If data1-4 indicates an error, data5-8 are also returned: | |
| data5-8: | Error code |

### 40.15 Get Track Token Type

*(PdrGetTrackTokenType)*

| **Command:** | | |
|---|---|---|
| data1-2: | Track token | |
| **Reply:** | | |
| data1: | Track type as follows: | |
| | 0: | Invalid track |
| | 1: | Video track |
| | 2: | Audio track |
| | 3: | Timecode track |
| | 4: | MPEG video track |
| | 5: | PdrDvcpro25Track |
| | 6: | PdrDvcpro50Track |

### 40.16 Get Track Type From Resource

*(PdrGetTrackTypeFromResourceType)*

| **Command:** | | |
|---|---|---|
| data1: | Resource type | |
| **Reply:** | | |
| data1: | Track type | See 40.13 for breakdown; 0 = invalid |
| If data1 is 0, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 40.17 Get Resource Type From Track Type

*(PdrGetResourceTypeFromTrackType)*

| **Command:** | |
|---|---|
| data1: | Track type (See **Get Track Token Type** results for values) |
| **Reply:** | |
| data1: | Resource type (See **Allocate Resource** for breakdown, 0x0b is invalid) |

### 40.18 Get Null Media Token

*(PdrNullMediaToken)*

| **Command:** |
|---|
| (No command data bytes required) |
| **Reply:** |
| data1-8: Null media token. |

### 40.19 Is Media Token Null

*(PdrMediaTokenIsNull)*

| **Command:** | | |
|---|---|---|
| data1-8: | Media token to test | |
| **Reply:** | | |
| data1: | 0: | True, it is a null media token. |
| | 1: | False, it is not a null media token. |

### 40.1A Get Media State

(*PdrGetMediaStateInfo*)

| **Command:** | | |
|---|---|---|
| data1-4: | Movie token | |
| data5-6: | Track token. (Two bytes only.) | |
| data7-14: | Reference media token | |
| data15: | Which media to get state info for: | |
| | 0: | Previous media segment |
| | 1: | This media segment |
| | 2 or greater | Next media segment |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| data2: | Media token type (PdrMediaTokenType): | |
| | 2: | PdrMediaFile |
| | 3: | PdrMediaBlack |
| data3: | Media attributes (PdrAttributes): | |
| | 1: | PdrReadOnly |
| | 2: | PdrControlRO |
| data4: | Track type (PdrTrackType): | |
| | 1: | PdrVideoTrack |
| | 2: | PdrAudioTrack |
| | 3: | PdrTimeCodeTrack |
| data5-8: | Media mark-in | |
| data9-12: | Media mark-out | |
| data13-16: | Media position | |
| data17-24: | Previous media token. (Null media token is 00 00 00 00 20 5A 00 00.) | |
| data25-32: | This media token | |
| data33-40: | Next media token | |
| data41-56: | data41-44: | rampUpGain |
| | data45-48: | rampDownGain |
| | data49-52: | rampUpFields |
| | data53-56: | rampDownFields |
| data57-(57+n): | Media file path name. (Null terminated.) | |
| If data1 indicates an error, data2-5 are returned: | | |
| data2-5: | Error code | |

### 40.1B Get Media State Info

*(PdrGetMediaStateInfo)*

| **Command:** | | |
|---|---|---|
| data1-4: | Movie token | |
| data5-6: | Track token. (Two bytes only.) | |
| data7-14: | Reference media token | |
| data15: | Which media to get state info for: | |
| | 0: | Previous media segment |
| | 1: | This media segment |
| | 2 or greater | Next media segment |
| data16-19: | The database version number | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| data2: | Media token type (PdrMediaTokenType): | |
| | 2: | PdrMediaFile |
| | 3: | PdrMediaBlack |
| data3: | Media attributes (PdrAttributes): | |
| | 1: | PdrReadOnly |
| | 2: | PdrControlRO |
| data4: | Track type (PdrTrackType): | |
| | 1: | PdrVideoTrack |
| | 2: | PdrAudioTrack |
| | 3: | PdrTimeCodeTrack |
| data5-8: | Media mark-in | |
| data9-12: | Media mark-out | |
| data13-16: | Media position | |
| data17-24: | Previous media token. (Null media token is 00 00 00 00 20 5A 00 00.) | |
| data25-32: | This media token | |
| data33-40: | Next media token | |

### 40.1B Get Media State Info (Continued) (*PdrGetMediaStateInfo*)

| | | |
|---|---|---|
| data41-56: | If data57 is FileTypeJpegVideo | |
| | data41-42: | MaxChrQ * 256 |
| | data43-44: | MaxLumQ * 256 |
| | data45-46: | MinChrQ * 256 |
| | data47-48: | MinLumQ * 256 |
| | data49-56: | 0 |
| | If data57 is FileTypeMpegVideo | |
| | data41: | Mpeg Chroma |
| | data42: | MpegGopEnd |
| | data43: | MpegPixStructure |
| | data44: | IPixPerGop |
| | data45: | PPixPerIPix |
| | data46: | BPixPerIPPix |
| | data47-56: | 0 |
| | If data57 is FileTypeAudio (values set by SetMediaFadesAndLevels) | |
| | data41-44: | rampUpGain |
| | data45-48: | rampDownGain |
| | data49-52: | rampUpFields |
| | data53-56: | rampDownFields |
| | If data57 is FileTypeTimecode | |
| | data41-44: | firstTimecodeValue |
| | data45-48: | lastTimecodValue |
| | data49-52: | timecodeFormat (0= dropFrame, else 1) |
| | data53-56: | 0 |
| data57: | FileType | |
| data58-59: | SampleRate | |
| data60-63: | SampleSize | |
| data64-64+N: | Media file path name. (Null terminated.) | |
| If data1 indicates an error, data2-5 are returned: | | |
| data2-5: | Error code | |

# Attached media access commands (0x50)

Attached media access commands are used to manipulate media on a timeline.

### 50.00 Get Attached Media Status

(bitmask for calls)

| **Command:** | | | |
|---|---|---|---|
| data1-4: | Handle of media for which status is needed | | |
| data5-6: | Mask indicating what status information is to be returned: | | |
| | bit0: | Starting position for media (field number on timeline) | (*VdrGetMediaStartPosition*) |
| | bit1: | Ending position for media (field number on timeline) | (*VdrGetMediaEndPosition*) |
| | bit2: | Mark-in point (fieldnumber within media) | (*VdrGetMediaMarkIn*) |
| | bit3: | Mark-out point (field number within media) | (*VdrGetMediaMarkOut*) |
| | bit4: | Media Name | (*VdrGetMediaName*) |
| | bit5: | Mark duration (Mark-out point minus Mark-in point) | (*VdrGetMediaMarkDuration*) |
| | bit6: | Media user data | (*VdrGetMediaUserData*) |
| | bit7: | Media duration on timeline | (*VdrGetMediaDuration*) |
| | bit8...bit15: | Reserved | |
| **Immediate Reply:** | | | |
| Ack + transaction number | | | |
| **Transaction Reply:** | | | |
| data1: | Transaction number | | |
| data2-3: | A copy of the request mask from command data5-6 (any request which is not implemented will cause the corresponding bit in the mask to be cleared) | | |
| data4-n: | Status values requested (value corresponding to the lowest bit in the mark first: | | |
| | Starting position for media (integer, 4 bytes) | | |
| | Ending position for media (integer, 4 bytes) | | |
| | Mark-in point for media (integer, 4 bytes) | | |
| | Mark-out point for media (integer, 4 bytes) | | |
| | Name of media (Null terminated string) | | |
| | Mark duration for media (integer, 4 bytes) | | |
| | Duration on timeline which media occupies (integer, 4 bytes) | | |
| | User data (integer, 4 bytes) | | |
| | Media duration (integer, 4 bytes) | | |

### 50.01 Attach Media

(*VdrAttachFittedMedia*)
(*VdrAttachMedia*)

| **Command:** | | |
|---|---|---|
| data1-n: | Name of media (Null terminated) | |
| data(n+1)-(n+4): | Handle of resource to which media is to attached | |
| data(n+5)-(n+8): | Optional number of fields duration over which the media should be played | |
| data(n+9)-(n+12): | Handle of media before which this media should be attached. Use value 0 to attach at end of list of media | |
| data(n+13): | Shift mode. One of the following: | |
| | 0: | Any media that were attached before the data(n+9)-(n+12) handle will be shifted to an earlier timeline position. |
| | 1: | Any media that were attached after and/or include the data(n+9)-(n+12) handle will be shifted to a later timeline position.<br>If the handle is Null, ShiftAfter has no effect on other media, while ShiftBefore effects *all* other media by moving them to an earlier position on the timeline. |

| **Immediate Reply:** | |
|---|---|
| Ack + transaction number | |

| **Transaction Reply:** | |
|---|---|
| data1: | Transaction number |
| data2-5: | Handle of attached media. (0 if media could not be attached.) |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |

**Comments:**

If there are no arguments after the resource handle, the media will be attached at the end of the list with a shift appropriate to the current position with duration of full media file length. Duration is 0, otherwise duration is the amount of time over which the media in/out is scheduled.

### 50.02 Attach Media With Marks

(*VdrAttachFittedMediaWithMarks*)
(*VdrAttachMediaWithMarks*)

| **Command:** | | |
|---|---|---|
| data1-n: | Name of media (null terminated) | |
| data(n+1)-(n+4): | Mark-in (integer) | |
| data(n+5)-(n+8): | Mark-out (integer) | |
| data(n+9)-(n+12): | Handle of resource to which media is to be attached. | |
| data(n+13)-(n+16): | Optional number of fields duration over which the media should be played | |
| data(n+17)-(n+20): | Handle of media before which this media should be attached. Use value 0 to attach at end of list of media | |
| data(n+21): | Shift mode. One of the following: | |
| | 0: | Any media that were attached before the data(n+17)-(n+20) handle will be shifted to an earlier timeline position. |
| | 1: | Any media that were attached after and/or include the datadata(n+17)-(n+20) handle will be shifted to a later timeline position. |
| | | If the handle is Null, ShiftAfter has no effect on other media, while ShiftBefore effects *all* other media by moving them to an earlier position on the timeline. |

| **Immediate Reply:** |
|---|
| Ack + transaction number |

| **Transaction Reply:** | |
|---|---|
| data1: | Transaction number |
| data2-5: | Handle of attached media. (0 if media could not be attached.) |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |

**Comments:**

If there are no arguments after the resource handle, the media will be attached at the end of the list with a shift appropriate to the current position with duration of full media file length. Duration is 0, otherwise duration is the amount of time over which the media in/out is scheduled.

### 50.03 Detach Media

*(VdrDetachMedia)*

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of media | |
| data5: | Shift mode. One of the following: | |
| | 0: | Any media that were attached before the data1-4 handle will be shifted to a later timeline position. |
| | 1: | Any media that were attached after the data1-4 handle will be shifted to an earlier timeline position. |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 50.04 Get Next Media

*(VdrGetNextMedia)*

| **Command:** | |
|---|---|
| data1-4: | Handle of media |
| data5-8: | Optional resource handle. Must be specified for media handle 0 usage. |
| **Reply:** | |
| data1-4: | Handle of the next media on the timeline Null indicates error |
| If data1-4 indicates an error, data5-8 are also returned: | |
| data5-8: | Error code |
| **Comments:** | |
| The reply contains the handle of the media following this media on the timeline. To get the handle of the first media on the timeline, use value 0. | |

### 50.05 Get Previous Media

*(VdrGetPreviousMedia)*

| **Command:** | |
|---|---|
| data1-4: | Handle of media |
| data5-8: | Optional resource handle. Must be specified for media handle 0 usage. |
| **Reply:** | |
| data1-4: | Handle of the previous media on the timeline Null indicates error |
| If data1-4 indicates an error, data5-8 are also returned: | |
| data5-8: | Error code |
| **Comments:** | |
| The reply contains the handle of the media preceding this media on the timeline. To get the handle of the last media on the timeline, use value 0. | |

### 50.06 Get Media at Position

(*VdrGetMediaAtPosition*)

| **Command:** | |
|---|---|
| data1-4: | Handle of resource to which media is attached |
| data5-8: | Position (field number) on timeline. |
| **Reply:** | |
| data1-4: | Handle of media at position on the timeline Null indicates error |
| If data1-4 indicates an error, data5-8 are also returned: | |
| data5-8: | Error code |
| **Comments:** | |
| The reply contains the handle of a media which has fields located at this position. (Null if none.) | |

### 50.07 Set Media Mark In

(*VdrSetFittedMediaMarkIn*)
(*VdrSetMediaMarkIn*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of media for which mark-in and mark-out are to be set. | |
| data5-8: | Mark-in point (field within media) | |
| data9: | Shift mode. One of the following: | |
| | 0: | Any media that were set for mark-in before the data5-8 handle will be shifted to an earlier timeline position. |
| | 1: | Any media that were set for mark-in after and/or including the data5-8 handle will be shifted to a later timeline position. |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comments:** | | |
| If there are no arguments after the mark-in point, the mark will be changed with a shift appropriate to the current position. | | |

### 50.08 Set Media Mark Out

(*VdrSetFittedMediaMarkOut*)
(*VdrSetMediaMarkOut*)

**Command:**

| | | |
|---|---|---|
| data1-4: | Handle of media for which mark-in and mark-out are to be set | |
| data5-8: | Mark-out point (field within media) | |
| data9: | Shift mode. One of the following: | |
| | 0: | Any media that were set for mark-out before the data5-8 handle will be shifted to an earlier timeline position. |
| | 1: | Any media that were set for mark-out after and/or including the data5-8 handle will be shifted to a later timeline position. |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

If there are no arguments after the mark-out point, the mark will be changed with a shift appropriate to the current position.

### 50.09 Set Media Duration

*(VdrSetMediaDuration)*

**Command:**

| | | |
|---|---|---|
| data1-4: | Handle of media for which the duration is to be set | |
| data5-8: | Duration (number of fields) | |
| data9: | Shift mode. One of the following: | |
| | 0: | Any media that were attached before the data5-8 handle will be shifted to an earlier timeline position. |
| | 1: | Any media that were attached after and/or including the data5-8 handle will be shifted to a later timeline position. |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

If there are no arguments after the duration, the mark will be changed with a shift appropriate to the current position.

### 50.0A Get Port Handle From Media Handle

*(VdrGetPortFromMedia)*

**Command:**

| | |
|---|---|
| data1-4: | Handle of media |

**Reply:**

| | |
|---|---|
| data1-4: | Handle for the port Null indicates error |

If data1-4 indicates an error, data5-8 are also returned:

| | |
|---|---|
| data5-8: | Error code |

### 50.0B Set Media User Data

*(VdrSetMediaUserData)*

**Command:**

| | | |
|---|---|---|
| data1-4: | Handle of media for which the user data is to be set | |
| data5-8: | User data (any 4 bytes) | |

**Reply:**

| | | |
|---|---|---|
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data1 indicates an error, data2-5 are also returned:

| | |
|---|---|
| data2-5: | Error code |

### 50.0C Get Resource From Media

(*VdrGetResourceFromMedia*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of media for which resource handle is desired | |
| **Reply:** | | |
| data1-4: | Handle of resource Null indicates error | |
| If data1-4 indicates an error, data5-8 are also returned: | | |
| data5-8: | Error code | |

### 50.0D Get Media Audio Fades

(*VdrGetMediaAudioFades*)

| **Command:** | | |
|---|---|---|
| data1-4: | MediaHandle | |
| **Reply:** | | |
| data1: | 0: | OK |
| | 1: | Error condition |
| data2-5: | up level (float) | GetLastError value |
| data6-9: | up fields | |
| data10-13: | down level (float) | |
| data14-17: | down fields | |

### 50.0E Set Media Audio Fades

(*VdrSetMediaAudioFades*)

| **Command:** | | |
|---|---|---|
| data1-4: | MediaHandle | |
| data5-8: | up level (float) | |
| data9-12: | up fields | |
| data13-16: | down level (float) | |
| data17-20: | down fields | |
| **Reply:** | | |
| data1: | 0: | OK |

**Comments:**

This function allows control of the audio media segment levels. It allows the cut between segments to be dropped to a very low level and the next segment playout level to be controlled.

### 50.0F Attach Media With Marks And Fades

(no API equivalent)

| **Command:** | |
|---|---|
| data1-n: | Name of media (Null terminated) |
| data(n+1)-(n+4) | Handle of resource to which media is to be attached |
| data(n+5)-(n+6): | 2-byte bit-mask specifying which optional parameters are present:<br>• bit0: Mark In and Mark Out values<br>• bit1: Duration<br>• bit2: Audio fade information<br>• bit3: Before handle<br>• bit4: Shift mode |

### 50.0F Attach Media With Marks And Fades

(no API equivalent)

| | |
|---|---|
| data(n+7)-m: | Optional parameters (4 bytes each). Only the parameters for which the bit is set in the bit-mask above can appear in this list of parameters. Those that are listed must follow this order:<br>• Mark In<br>• Mark Out<br>• Duration (Number of fields duration over which the media should be played)<br>• Up Level (float)<br>• Up Fields<br>• Down Level (float)<br>• Down Fields<br>• Before media handle (Handle of media before which this media should be attached. Use 0 or omit this parameter to attach at end of list of media.)<br>• Shift mode (See Attach Media for description) |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | |
|---|---|
| data1: | Transaction number |
| data2-5: | Handle of attached media (0 if media could not be attached) |

If data2-5 indicates an error, data6-9 are also returned:

| | |
|---|---|
| data6-9: | Error code |

**Comments:**

This function performs the equivalent of Attach Media [With Marks] (50.01 or 50.02) followed by Set Media Audio Fades (50.0E).

### 50.10 Attach Media By Movie With Marks And Fades

(no API equivalent)

**Command:**

| | | |
|---|---|---|
| data1-n: | Name of movie (Null terminated) | |
| data(n+1): | Track type (see PdrGetTrackTokenType) | |
| data(n+2): | Track count. Zero-based value indicating which track of the given type to use. | |
| data(n+3): | One of the following: | |
| | 0 | The value in data(n+4)-(n+7) is to be used as an ordinal number (starting with 1) of the media file within the specified track of the movie. |
| | 1 | The value in data(n+4)-(n+7) is to be used as a timeline position |
| data(n+4)-(n+7): | Ordinal number or timeline position indicating which media within the given track to attach. | |
| data(n+8)-(n+11): | Handle of resource to which media is to be attached. | |
| data(n+12)-(n+13): | 2-byte bit-mask specifying which optional parameters are present: | |
| | bit0: | Mark In and Mark Out values |
| | bit1: | Duration |
| | bit2: | Audio fade information |
| | bit3: | Before handle |

### 50.10 Attach Media By Movie With Marks And Fades

(no API equivalent)

| | |
|---|---|
| | bit4: Shift mode |
| data(n+14)-m: | Optional parameters (4 bytes each). Only the parameters for which the bit is set in the bit-mask above can appear in this list of parameters. Those that are listed must follow this order:<br>• Mark In<br>• Mark Out<br>• Duration (Number of fields duration over which the media should be played)<br>• Up Level (float)<br>• Up Fields<br>• Down Level (float)<br>• Down Fields<br>• Before media handle (Handle of media before which this media should be attached. Use 0 or omit this parameter to attach at end of list of media.)<br>• Shift mode (See Attach Media for description) |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | |
|---|---|
| data1: | Transaction number |
| data2-5: | Handle of attached media (0 if media could not be attached) |

If data2-5 indicates an error, data6-9 are also returned:

| | |
|---|---|
| data6-9: | Error code |

**Comments:**

The first 5 parameters define which media file will be attached by this command. The given movie will be searched for a track of the given type. The track count then defines which track of the given type will be searched (For example, if track count is 0, the first track of the given type will be used; if track count is 1, the second track of the given type will be used). Once the proper track is determined, the next two parameters will be used to decide which media file within the track to use for the attach. If data(n+3) is 0, the next parameter will specify the ordinal position of the media to use (1 means the first media on the track, 2 means the second media file on the track, etc.). If data(n+3) is 1, the next parameter will specify a timeline position. The media file that would be at the given timeline position if the track were attached starting at timeline position 0 is the media that will be attached by this command.

# Resource access commands (0x60)

Resource access commands are used to:

- Allocate, connect, and release physical resources to ports
- Set and retrieve the operating parameters of these resources

Parameters include luminance and chrominance Q for video codecs, audio input/output levels, and timecode generator formats and modes.

### 60.00 Get Resource Status

(bitmask for calls)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of resource for which status is needed | |
| data5-6: | Mask indicating what status information is to be returned: | |
| | For JPEG codecs only: | |
| | bit0: Current field size in bytes | (*VdrGetCurrentFieldSize*) |
| | bit1: Current luminance Q factor | (*VdrGetCurrentLumQFactor*) |
| | bit2-15: Reserved | |
| | For MPEG codecs only: | |
| | Current frame size (integer, 4 bytes) | |
| | Current pix coding (integer, 1 byte) | |
| | 1: I-frame | |
| | 2: P-frame | |
| | 3: B-frame | |
| | Current pix structure (integer, 1 byte) | |
| | 1: topfield | Only “frame” is supposed/ returned at this time. |
| | 2: bottomfield | |
| | 3: frame | |
| | For DVCPRO codecs only: | |
| | bit0: Current frame size (integer, 4 bytes) | |
| | For audio codecs only: | |
| | bit0: Current audio level | (*VdrGetAudioLevel*) |
| | bit1-15: Reserved | |
| | For timecode recorders/generators only: | |
| | bit0: Current timecode | (*VdrGetCurrentTimeCode*) |
| | bit1: Current user bits | (*VdrGetCurrentUserBits*) |
| | bit2-15: Reserved | |

### 60.00 Get Resource Status

(bitmask for calls)

**Reply:**

data1-2: A copy of the request mask from command data5-6 (any request not implemented causes the corresponding mask bit to be cleared)

data3-n: Status values requested (value corresponds to the lowest bit in the mask first):

For video codecs only:

Current field size (integer, 4 bytes)

Current luminance Q factor (float, 4 bytes)

For audio codecs only:

Current audio level (integer, 4 bytes)

For timecode recorders/generators only:

Current timecode (timecode, 4 bytes)

For this command, timecodes are represented as:

| | |
|---|---|
| byte0 | Frames (BCD, with bit 6 indicating drop-frame when set) |
| byte1 | Seconds (BCD) |
| byte2 | Minutes (BCD) |
| byte3 | Hours (BCD) |

Current user bits (integer, 4 bytes)

**Comments:**

This command returns the status of the specified resource.

### 60.01 Get Resource Configuration

(bitmask for calls)

| **Command:** | | | |
|---|---|---|---|
| data1-4: | Handle of resource for which status is needed | | |
| data5-6: | Mask indicating what status information is to be returned: | | |
| | bit0: | resource type | (*VdrGetResourceType*) |
| | bit1: | resource number | (*VdrGetResourceNumber*) |
| | bit2: | is resource active? | (*VdrIsResourceActive*) |
| | bit8-15: | reserved | |
| | For video codecs only: | | |
| | bit3: | goal size | (*VdrGetVideoGoalSize*) |
| | bit4: | minimum luminance Q | (*VdrGetMinLumQ*) |
| | bit5: | maximum luminance Q | (*VdrGetMaxLumQ*) |
| | bit6: | minimum chrominance Q | (*VdrGetMinChrQ*) |
| | bit7: | maximum chrominance Q | (*VdrGetMaxChrQ*) |
| | bit8: | minimum luminance Q [a] | (*VdrGetAbsMinLumQ*) |
| | bit9: | maximum luminance Q [a] | (*VdrGetAbsMaxLumQ*) |
| | bita: | minimum chrominance Q [a] | (*VdrGetAbsMinChrQ*) |
| | bitb: | maximum chrominance Q [a] | (*VdrGetAbsMaxChrQ*) |
| | For JPEG or DVCPRO codecs or MPEG decoder pairs: | | |
| | bitc: | video codec timing | (*VdrGetVideoCodecTiming*) |
| | For JPEG or DVCPRO codecs or MPEG encoders pairs: | | |
| | bitd | Bit rate | (*VdrGetBitRate*) |
| | For MPEG encoders: | | |
| | bit4: | MPEG chroma format | (*VdrGetMpegChromaFormat*) |
| | bit5: | MPEG GOP structure | (*VdrGetMpegGopStructure*) |
| | bit6: | MPEG encoding reange | (*VdrGetEncodingRange*) |
| | bit7 | Saved VBI lines | (*VdrGetSavedVbiLines*) |
| | For audio codecs only: | | |
| | bit3: | input level | (*VdrGetAudioInputLevel*) |
| | bit4: | output level | (*VdrGetAudioOutputLevel*) |
| | bit5: | audio jog window length | (*VdrGetAudioWindow*) |
| | For time-code generators only: | | |
| | bit3: | generator mode | (*VdrGetGenTcMode*) |
| | bit4: | generator format | (*VdrGetGenTcFormat*) |

### 60.01 Get Resource Configuration (Continued)

(bitmask for calls)

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | |
|---|---|
| data1: | Transaction number |
| data2-3: | A copy of the request mask from command data5-6 (any request not implemented causes the corresponding bit in the mask to be cleared) |
| data4-n: | Status values requested (value corresponding to the lowest bit in the mask first: |
| | Resource type (one byte; see 60.02 *Allocate Resource*) |
| | Resource number (one byte): the logical number of the resource within each resource type |
| | Resource-active indication (one byte): |
| | 0: inactive |
| | 1: active |
| | For video codecs only: |
| | goal size (integer, 4 bytes) |
| | minimum luminance Q (float, 4 bytes) |
| | maximum luminance Q (float, 4 bytes) |
| | minimum chrominance Q (float, 4 bytes) |
| | maximum chrominance Q (float, 4 bytes) |
| | For JPEG or DVCPRO eodecs or MPEG decoder pairs: |
| | video codec timing (one byte) |
| | For MPEG encoders: |
| | MPEG chroma (one byte) |
| | MPEG GOP structure (five one-byte values) |
| | mpegGopEnd |
| | iPixPerGop |
| | pPixPerIPix |
| | bPixPerIPPix |
| | pixStructure |
| | number of luma lines saved (1 byte) |
| | list of luma lines saved (2 bytes each) |
| | number of chroma lines saved (1 byte) |
| | list of chroma lines saved (2 bytes each) |
| | For JPEG or DVCPRO eodecs or MPEG encoders: |
| | bit-rate (double, 8 bytes) |

### 60.01 Get Resource Configuration (Continued)

(bitmask for calls)

For audio codecs only:

input level; 1.0 indicates unity gain, or 0dB (float, 4 bytes)

output level; 1.0 indicates unity gain, or 0dB (float, 4 bytes)

audio jog window length, in fields (integer, 4 bytes)

For time-code generators only:

Generator mode (one byte). One of the following:

| | |
|---|---|
| 0: | Freeze; generator value does not change |
| 1: | Free-run; generator value advances with time, even when position along timeline is unchanged |
| 2: | Field-locked; generator value changes as position along timeline changes |

Generator format (one byte). One of the following:

| | |
|---|---|
| 0: | DropFrame |
| 1: | NonDropFrame |

**Comments:**

This command returns the configuration of the current resource. Resource type is the type of the specified resource. Resource number is the number of the resource.

For a video codec, goal size is for video compression (record only), minimum and maximum luminance Q are the minimum and maximum luminance quantization levels for the codec, and minimum and maximum chrominance Q are the minimum and maximum chrominance quantization levels.

For an audio codec, input level and output level are the current audio input and output level settings. Audio jog window length is the repetition level interval (in fields) when audio playback occurs during still play mode.

a. Absolute system value.

### 60.02 Allocate Resource

(*VdrAllocateResource*)

| **Command:** | | |
|---|---|---|
| data1-4: | Port handle | |
| data5: | Type of resource to be allocated. One of the following: | |
| | 0: | audio codec |
| | 1: | JPEG codec |
| | 2: | video input |
| | 3: | video output |
| | 4: | LTC input |
| | 5: | VITC input |
| | 6: | timecode generator |
| | 7: | LTC output |
| | 8: | VITC output |
| | 9: | timecode recorder |
| | a: | audio output |
| | b: | video combiner |
| | c: | MPEG standard definition encoder |
| | d: | MPEG standard definition decoder |
| | e: | audio input |
| | f: | DVCPRO25 codec |
| | 10: | DVCPRO50 codec |
| | 11: | SDTI codec |
| | 12: | video codec (for use only with 00.18 Get Attributes by Type) |
| | 13 | MPEG high definition encoder |
| | 14: | MPEG high definition decoder |
| | 1b: | GPI input |
| | 1c: | GPI output |
| | 22: | MPEG standard definition codec (PVS1100 only) |
| data6: | Logical number of resource to be allocated | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2-5: | Handle to allocated resource 0 if resource could not be allocated | |
| If data2-5 is 0, data6-9 are also returned: | | |
| data6-9: | Error code | |

### 60.02 Allocate Resource (Continued)

(*VdrAllocateResource*)

**Comments:**

This command gains control of the specified physical resource.

When two concurrently running processes or threads attempt to use the **Allocate Resource** call, a lockup is possible. To avoid a lockup, always claim resources in this order:

- Codec resource (low to high codec number)
- Output resources (low to high channel number)
- Audio resources (low to high channel number)

It is best to claim all known needed resources before using any. If an **Allocate Resource** call fails, release all resources by using **Release Resource** calls. The process should then sleep 10 to 100 ms. (random sleep times in this range are best), before again attempting to claim resources.

### 60.03 Get Resource Connect Handle

*(VdrGetResourceConnectionHandle)*

| **Command:** | | |
|---|---|---|
| data1-4: | Port handle | |
| data5: | Type of resource to be connected. One of the following: | |
| | 0: | audio codec |
| | 1: | JPEG codec |
| | 2: | video input |
| | 3: | video output |
| | 4: | LTC input |
| | 5: | VITC input |
| | 6: | timecode generator |
| | 7: | LTC output |
| | 8: | VITC output |
| | 9: | timecode recorder |
| | a: | audio output |
| | b: | video combiner |
| | c: | MPEG standard definition encoder |
| | d: | MPEG standard definition decoder |
| | e: | audio input |
| | f: | DVCPRO25 codec |
| | 10: | DVCPRO50 codec |
| | 11: | SDTI codec |
| | 13 | MPEG high definition encoder |
| | 14: | MPEG high definition decoder |
| | 1b: | GPI input |
| | 1c: | GPI output |
| | 22: | MPEG standard definition codec (PVS 1100 only) |
| data6: | Logical number of resource to be connected | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2-5: | Connect handle to resource 0 if resource has no connection capability | |
| If data2-5 is 0, data6-9 are also returned: | | |
| data6-9: | Error code | |

### 60.03 Get Resource Connect Handle

*(VdrGetResourceConnectionHandle)*

**Comments:**

This command returns a handle that can be used for resource connection. While an allocated resource allows the program to control the resource, a connection handle to a resource is a shared access to the resource. This allows the resource to be connected into the configuration but does not allow control.

The handle does not allow resource control or modification. Output Resource must be allocated and cannot be connected.

### 60.04 Release Resource

*(VdrReleaseResource)*

**Command:**

| | |
|---|---|
| data1-4: | Handle of resource to be released |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command releases control of the specified resource. The resource handle cannot be used again after this call.

A **Close Port** call releases all resource handles obtained using **Allocate Resource** for that port. A **Release Resource** call detaches all media handles obtained using **Attach Media** for the resource. The resource and media handles cannot be used after these calls.

### 60.05 Reset Resource

*(VdrResetResource)*

**Command:**

| | |
|---|---|
| data1-4: | Handle of resource to be reset |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command resets the specified resource. Resetting a resource detaches all media attached to that resource.

### 60.06 Get Next Resource

(*VdrGetNextResource*)
(*VdrGetNextResourceOfType*)

| **Command:** | |
|---|---|
| data1-4: | Handle of port |
| data5-8: | Handle of reference resource (0 if reference resource is the last resource) |
| data9: | Type of resource (see 60.02 *Allocate Resource* for details) |
| **Reply:** | |
| data1-4: | Handle of the next resource |
| data5: | Type of resource (see 60.02 *Allocate Resource* for details) |
| data6: | Logical number of resource (see 60.02 *Allocate Resource* for details) |

**Comments:**

This command locates the next resource handle attached to the specified port handle.

The reply contains the handle of the resource following the reference resource. To get the handle of the first resource, use value 0. If there are no arguments after the resource handle, the handle to the next resource of any type is returned.

### 60.07 Get Previous Resource

(*VdrGetPreviousResource*)
(*VdrGetPreviousResourceOfType*)

| **Command:** | |
|---|---|
| data1-4: | Port handle |
| data5-8: | Reference resource handle |
| data9: | Type of resource (see 60.02 *Allocate Resource* for details) |
| **Reply:** | |
| data1-4: | Handle of the previous resource (0 if reference resource handle is the first resource) |
| data5: | Type of resource (see 60.02 *Allocate Resource* for details) |
| data6: | Logical number of resource (see 60.02 *Allocate Resource* for details) |

**Comments:**

This command locates the previous resource handle attached to the specified port handle.

The reply contains the handle of the preceding resource. To get the handle of the last resource, use 0. If there are no arguments after the resource handle, the handle to the next resource of any type is returned.

### 60.08 Activate Resource

(*VdrActivateResource*)

**Command:**

| | |
|---|---|
| data1-4: | Handle of resource to be activated |

Multiple resources can be activated by providing their handles in this list, for example:

| | |
|---|---|
| data5-8: | Handle of resource (2) to be activated |
| data9-12: | Handle of resource (3) to be activated |
| ... | |
| data(n)-(n+3): | Handle of resource to be activated |

A maximum of 32 resources can be in the list.

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Summary error code. One of the following: | |
| | 0: | OK |
| | 1: | Error condition |
| data3-n: | One of the following per resource handle in the command: | |
| | 0: | OK |
| | 1: | Error condition |

**Comments:**

This command activates the specified resource.

Active resources can play/record material. Inactive audio resources produce silence. Output from inactive video resources is undefined. The default state of a resource is active after receiving the handle from the **Allocate Resource** call.

### 60.09 Deactivate Resource

*(VdrDeactivateResource)*

**Command:**

| | |
|---|---|
| data1-4: | Handle of resource to be deactivated |

Multiple resource can be deactivated by providing their handles in this list, for example:

| | |
|---|---|
| data5-8: | Handle of resource (2) to be deactivated |
| data9-12: | Handle of resource (3) to be deactivated |
| ... | |
| data(n)-(n+3): | Handle of resource to be deactivated |

A maximum of 32 resources can be deactivated with a single list.

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Summary error code. One of the following: | |
| | 0: | OK |
| | 1: | Error condition |
| data3-n: | One of the following per resource handle in the command: | |
| | 0: | OK |
| | 1: | Error condition |

**Comments:**

This command deactivates the specified resource.

Inactive resources neither play nor record material (on playback, video codecs play black, audio codecs play silence, and timecode recorders play 00:00:00:00).

### 60.0A Set Video Goal Size

*(VdrSetVideoGoalSize)*

**Command:**

| | |
|---|---|
| data1-4: | Handle of video codec resource. The goal size (in bytes/field) is set for this resource |
| data5-8: | Goal size, in bytes/field (integer) |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command sets the video compression goal size (record only) for the specified video codec resource.

The specified goal size affects all video media files recorded by the resource. The default is 75000.

### 60.0B Set Luminance Q Limits

*(VdrSetLumQLimits)*

**Command:**

| | |
|---|---|
| data1-4: | Handle of video codec resource (luminance Q limits will be set for this resource) |
| data5-8: | Minimum luminance Q (float) |
| data9-12: | Maximum luminance Q (float) |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command sets the minimum and maximum luminance Q limits for the specified video codec.

### 60.0C Set Chrominance Q Limits

(*VdrSetChrQLimits*)

| **Command:** | |
|---|---|
| data1-4: | Handle of video codec resource. The chrominance Q limits will be set for this resource |
| data5-8: | Minimum chrominance Q (float) |
| data9-12: | Maximum chrominance Q (float) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |
| **Comments:** | |
| This command sets the minimum and maximum chrominance Q limits for the specified video codec. | |

### 60.0D Set Audio Input Level

(*VdrSetAudioInputLevel*)

| **Command:** | |
|---|---|
| data1-4: | Handle of audio codec resource |
| data5-8: | Audio input level (float); 1.0 indicates unity gain (0dB, maximum setting) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |
| **Comments:** | |
| This command sets the level between the audio input (the rear panel connection) and the audio codec. It sets the input level (which determines recorded material loudness and the loudness of material in E-E mode) for the specified resource. | |

### 60.0E Set Audio Output Level

*(VdrSetAudioOutputLevel)*

**Command:**

| | |
|---|---|
| data1-4: | Handle of audio codec resource |
| data5-8: | Audio output (maximum setting) level (float); 1.0 indicates unity gain (0dB) |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command sets the level between the audio codec and the audio output (the rear panel connection). It sets the playback loudness of previously recorded material for the specified resource.

### 60.0F Set Audio Window

*(VdrSetAudioWindow)*

**Command:**

| | |
|---|---|
| data1-4: | Handle of audio codec resource. The audio window (which determines how many fields of audio will be played during each jog in mode) will be set for this resource |
| data5-8: | Audio window (integer) |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command sets the repetition interval (in fields) for the specified audio resource when audio playback occurs during still play mode. During still play, the leakage frequency that appears in the audio is inversely related to the window size. This affects all attached audio media files for the specified audio resource. This call is volatile. The default is 1.

### 60.10 Set Timecode Generator Format

*(VdrSetGenTcFormat)*

**Command:**

| | | |
|---|---|---|
| data1-4: | Handle of timecode generator resource | |
| data 5: | Timecode format. One of the following: | |
| | 0: | Drop-frame (60 Hz video format only) |
| | 1: | Non-drop frame |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command sets the timecode format for the specified time-code generator.

### 60.11 Set Timecode Generator Mode

*(VdrSetGenTcMode)*

**Command:**

| | | |
|---|---|---|
| data1-4: | Handle of timecode generator resource | |
| data 5: | Time-code generator mode. One of the following: | |
| | 0: | Freeze; generator value does not change |
| | 1: | Free-run; generator value advances with time, even when timeline position is unchanged |
| | 2: | Field-locked; generator value changes as position along timeline changes |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

This command sets the operating mode for the specified time-code generator.

### 60.12 Is Resource Available

(*VdrIsResourceAvailable*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| data5: | Type of resource (see 60.02 *Allocate Resource* for details) | |
| data6: | Number of resource (see 60.02 *Allocate Resource* for details) | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK (Resource is available) |
| | 1: | Error condition |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

### 60.13 Get Port Handle from Resource Handle

(*VdrGetPortFromResource*)

| **Command:** | |
|---|---|
| data1-4: | Handle of resource |
| **Reply:** | |
| data1-4: | Handle for the port Null indicates error |
| If data1-4 indicates an error, data5-8 are also returned: | |
| data5-8: | Error code |

### 60.14 Clear Timeline

convenience routine:
(*VdrCancelEvent*)
(*VdrDetachMedia*)
(*VdrDetachMovie*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of port | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 60.15 Set Resource User Data

(*VdrSetResourceUserData*)

| **Command:** | |
|---|---|
| data1-4: | Handle of resource |
| data5-8: | User data (any 4 bytes) |
| **Reply:** | |
| data1: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data1 indicates an error, data2-5 are also returned: | |
| data2-5: | Error code |

### 60.16 Get Resource User Data

(*VdrGetResourceUserData*)

| **Command:** | |
|---|---|
| data1-4: | Handle of resource |
| **Reply:** | |
| data1: | User data |

### 60.17 Get Video Combiner Part Handle

(*VdrGetVideoCombinerPartHandle*)

| **Command:** | |
|---|---|
| data1-4: | Handle of resource |
| data5: | Part Id (One of the enumerated values in **VdrGetVideoCombinerPartHandle**) |
| **Reply:** | |
| data1-4: | Resource handle of VC part Null indicates error |
| If data1-4 indicates an error, data5-8 are also returned: | |
| data5-8: | Error code |

### 60.18 Set Video Codec Timing

(*VdrSetVideoCodecTiming*)

*Engineering specification not yet available.*

### 60.19 Set Mpeg Chroma Format

(*VdrSetMpegChromaFormat*)

**Command:**

| | |
|---|---|
| data1-4: | Resource handle to MpegEncoder |
| data5: | MpegChroma |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

### 60.1A Set Mpeg Gop Structure

(*VdrSetMpegGopStructure*)

**Command:**

| | |
|---|---|
| data1-4: | Resource Handle to MpegEncoder |
| data5: | MpegGopEnd |
| data6: | iPixPerGop |
| data7: | pPixPerIPix |
| data8: | bPixPerIPPix |
| data9: | MpegPixStructure |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

### 60.1B Merge Resources

(*VdrMergeResources*)

*Engineering specification not yet available.*

### 60.1C Get Abs Port Config

(no API equivalent)

*Engineering specification not yet available.*

### 60.1D Set Bit-Rate

(*VdrSetBitRate*)

**Command:**

| | |
|---|---|
| data1-4: | Resource Handle to MpegEncoder |

### 60.1D Set Bit-Rate (*VdrSetBitRate*)

| | |
|---|---|
| data5-8: | Desired bitrate (float) |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

### 60.1E Get Mpeg Gop Limits (*VdrGetMpegGopLimits*)

*Engineering specification not yet available.*

### 60.1F Set Encoding Range (*VdrSetEncodingRange*)

*Engineering specification not yet available.*

### 60.20 Set Gen Tc Regen Source (*VdrSetGenTcRegenSource*)

**Command:**

| | |
|---|---|
| data1-4: | Resource handle to timecode generator. |
| data5-8: | Resource handle to timecode source. |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

### 60.21 Set Saved Vbi Lines (*VdrSetSavedVbiLines*))

**Command:**

| | |
|---|---|
| data1-4: | Resource handle to MPEG encoder |
| data5 | Number of lines of luminance data to be saved, equivalent to the number of entries in data6-m. |
| data6-m | List of 2-byte video line numbers for which lumniance data will be saved. |
| data(m+1) | Number of lines of chrominance data to be saved, equivalent to the number of entries in data(m+2)-data n. |

### 60.21 Set Saved Vbi Lines

(*VdrSetSavedVbiLines*))

| | |
|---|---|
| data(m+2)-n: | List of 2-byte video line numbers for which chrominance data will be saved. |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

# Event access commands (0x70)

(These commands specify how resources are connected and scheduled in the system.

### 70.00 Default Event

*(VdrDefaultEvent)*

**Command:**

| | | |
|---|---|---|
| data1-4: | Port handle | |
| data5: | Event type, specified as follows: | |
| | 2: | Mix audio |
| | 3: | Connect resources |
| | 4: | Set timecode generator time |
| | 5: | Set timecode generator user bits |
| data6-9: | Event handle or source-resource handle | |

The following depends on the type of event, as follows:

| | | |
|---|---|---|
| data10-13: | Connect resources: destination resource handle | |
| | Mix audio: | destination resource handle |
| | Set TC Gen Time: | timecode |
| | Set TC Gen Bits: | user bits pattern |

For event type Mixed audio, include the following data items:

| | |
|---|---|
| data14-17: | Mix level (float) (0.0...1.0) |
| data18-21: | Number of fields for duration of mix (integer) |

For event type Set timecode generator time, include the following data items:

| | |
|---|---|
| data14-17: | Field number at which to activate |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comments:**

Specifies an event that is effective during idle, record, or record-cue modes.

### 70.01 Schedule Event

*(VdrScheduleEvent)*

| **Command:** | | |
|---|---|---|
| data1-4: | Port handle | |
| data5-8: | Field number for event timing | |
| data9: | Event type, specified as follows: | |
| | 2: | Mix audio |
| | 3: | Connect resources |
| | 4: | Set timecode generator time |
| | 5: | Set timecode generator user bits |
| data10-13: | Source-resource handle | |
| The following depends on the type of event, as follows: | | |
| data14-17: | Connect resources: | destination resource handle |
| | Mix audio: | destination resource handle |
| | Set TC Gen Time: | timecode |
| | Set TC Gen Bits: | user bits pattern |
| For event type mixed audio, include the following data items: | | |
| data18-21: | Mix level (float) (0.0...1.0) | |
| data22-25: | Number of fields for duration of mix (integer) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2-5: | Event handle Null indicates error | |
| If data2-5 indicates an error, data6-9 are also returned: | | |
| data6-9: | Error code | |
| **Comments:** | | |

This command schedules the specified event, starting at the specified field number. Scheduled events are used to specify playback connections and conditions.

Reverse play causes the specified event to be reversed, restoring conditions in effect prior to the event. An attempt to schedule an event that overlaps another event will fail.

### 70.02 Reschedule Event

(*VdrRescheduleEvent*)

| **Command:** | | |
|---|---|---|
| data1-4: | Field number for the event timing | |
| data5-8: | Event handle | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comments:** | | |
| This command reschedules the previously scheduled event to occur at the specified field number. | | |

### 70.03 (Unused)

### 70.04 Cancel Event

(*VdrCancelEvent*)

| **Command:** | | |
|---|---|---|
| data1-4: | Handle of event to cancel | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comments:** | | |
| This command cancels the previously scheduled event. | | |

### 70.05 When Is Event

*(VdrWhenIsEvent)*

| **Command:** | |
|---|---|
| data1-4: | Handle of event to determine place on timeline |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Field number when event is scheduled (Null if error or timeline position is 0) |
| **Comments:** | |
| This command returns the timeline field number where the specified event is scheduled. | |

### 70.06 (Reserved)

### 70.07 (Reserved)

### 70.08 Get Port Handle from Event Handle

*(VdrGetPortFromEvent)*

| **Command:** | |
|---|---|
| data1-4: | Handle of event |
| **Reply:** | |
| data1-4: | Handle for the port. Null indicates error. |
| If data1-4 indicates an error, data5-8 are also returned: | |
| data5-8: | Error code |

### 70.09 Get Next Event Handle

*(VdrGetNextEvent)*

| **Command:** | |
|---|---|
| data1-4: | Handle of port |
| data5-8: | Handle of event from which to get the next event. (Null, to get first.) |
| **Reply:** | |
| data1-4: | Handle of the next event. Null indicates end. |
| If data1-4 is Null, data5-8 are also returned: | |
| data5-8: | Error code |

### 70.0A Get Previous Event Handle

*(VdrGetPreviousEvent)*

| **Command:** | |
|---|---|
| data1-4: | Handle of port |
| data5-8: | Handle of event from which to get the next event. (Null, to get last.) |
| **Reply:** | |
| data1-4: | Handle of the previous event. Null indicates first. |
| If data1-4 is Null, data5-8 are also returned: | |
| data5-8: | Error code |

### 70.0B Default Event At Gen Tc

(*VdrStateEventAtGenTc*)

*Engineering specification not yet available.*

### 70.0C State Event

(*VdrStateEvent*)

**Command:**

| | | |
|---|---|---|
| data1-4: | Port handle | |
| data5: | State mask that is the sum of any of the following: | |
| | 1: | Idle/Record |
| | 2: | Play Active |
| | 3: | Play Ready |
| data6: | Event type, specified as follows: | |
| | 1: | Mix Audio |
| | 2: | Connect Resources |
| | 3: | Set time code generator time |
| | 4: | Set time code generator bits |
| data6-9: | Resource Handle for source. | |

The following parameters depend on the type of the event as follows:

| | | |
|---|---|---|
| data10-13: | Connect Resources: | Destination Resource Handle |
| | Mix Audio: | Destination Resource Handle |
| | Set TC Gen Time: | Time code value. |
| | Set TC Gen Bits: | User bits pattern. |

For event type Mix Audio, include the following data items:

| | |
|---|---|
| data14-17: | Mix level (float) (0.0…1.0) |
| data18-21: | Number of fields for duration of mix (integer). |

For event type Set time code generator time, include the following item:

| | |
|---|---|
| Data14-17: | Field number at which to activate |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Error code | |
| | 0: | OK |
| | 1: | Error condition |

**Comments:**

Specifies an event and the Port modes when the event will be effective.

### 70.0D State Event At Gen Tc

(*VdrGetNextEvent*)

**Command:**

| | |
|---|---|
| data1-4: | Port handle |

### 70.0D State Event At Gen Tc

(*VdrGetNextEvent*)

| | | |
|---|---|---|
| data5-8: | Time code generator Resource Handle | |
| data9-12: | Time code value at which to trigger this State Event | |
| data13: | State mask that is the sum of any of the following: | |
| | 1: | Idle/Record |
| | 2: | Play Active |
| | 3: | Play Ready |
| data14: | Event type, specified as follows: | |
| | 2: | Mix Audio |
| | 3: | Connect Resources |
| | 4: | Set time code generator time |
| | 5: | Set time code generator bits |
| data15-18: | Resource Handle for source. | |

The following parameters depend on the type of the event as follows:

| | | |
|---|---|---|
| data10-13: | Connect Resources: | |
| | Mix Audio: | |
| | Set TC Gen Time: | |
| | Set TC Gen Bits: | |

For event type Mix Audio, include the following data items:

| | | |
|---|---|---|
| data14-17: | Mix level (float) (0.0…1.0) | |
| data18-21: | Number of fields for duration of mix (integer). | |

For event type Set time code generator time, include the following item:

| | | |
|---|---|---|
| Data14-17: | Field number at which to activate | |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Error code | |
| | 0: | |
| | 1: | |

### 70.0D State Event At Gen Tc

(*VdrGetNextEvent*)

| **Comments:** | |
|---|---|
| **Command:** | |
| data1-4: | Port handle |
| data5: | State mask that is the sum of any of the following: |
| | 1: |
| | 2: |
| | 3: |
| data6: | Event type, specified as follows: |

# Video mix effects access commands (0x80)

### 80.00 Allocate Context

*(VmeAllocateContext)*

| **Command:** | |
|---|---|
| data1: | Resource type |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Vme handle. Null indicates error |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |

### 80.01 Free Context

*(VmeFreeContext)*

| **Command:** | | |
|---|---|---|
| data1-4: | Vme handle | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 80.02 Apply Context

*(VmeApplyContext)*

| **Command:** | | |
|---|---|---|
| data1-4: | Resource handle from allocation of ResourceVideoCombiner | |
| data5-8: | Vme handle | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 80.03 Set Transition Type

*(VmeSetTransitionType)*

**Command:**

| | |
|---|---|
| data1-4: | Vme handle |
| data5: | Stage (one of enumerated values) |
| data6: | Transition type (one of enumerated values) |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comment:**

Enumerated values are described under **VmeSetTransitionType**.

### 80.04 Reset Transition Type

*(VmeResetTransition)*

**Command:**

| | |
|---|---|
| data1-4: | Vme handle |
| data5: | Stage (one of enumerated values) |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comment:**

Enumerated values are described under **VmeResetTransition**.

### 80.05 Start Transition

*(VmeStartTransition)*

| **Command:** | | |
|---|---|---|
| data1-4: | Resource handle from allocation of ResourceVideoCombiner | |
| data5: | Stage (one of enumerated values) | |
| data6: | Transition direction (one of enumerated values) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comment:** | | |
| Enumerated values are described under **VmeStartTransition**. | | |

### 80.06 Set Transition Progress

*(VmeSetTransitionProgress)*

| **Command:** | | |
|---|---|---|
| data1-4: | Resource handle from allocation of ResourceVideoCombiner | |
| data5: | Stage (one of enumerated values) | |
| data6-9: | Progress (0 through VME_MAX_PROGRESS) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comment:** | | |
| Enumerated values are described under **VmeSetTransitionProgress**. | | |

### 80.07 Get Transition Progress

*(VmeGetTransitionProgress)*

| **Command:** | |
|---|---|
| data1-4: | Resource handle from allocation of ResourceVideoCombiner |
| data5: | Stage (one of enumerated values) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | Progress (0 through VME_MAX_PROGRESS) Negative number indicates error |
| If data2-5 indicates an error, data6-9 are also returned: | |
| data6-9: | Error code |
| **Comment:** | |
| Enumerated values are described under **VmeGetTransitionProgress**. | |

### 80.08 Reset Fade

*(VmeResetFade)*

| **Command:** | | |
|---|---|---|
| data1-4: | Vme handle | |
| data5: | Stage (one of enumerated values) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comment:** | | |
| Enumerated values are described under **VmeResetFade**. | | |

### 80.09 Start Fade

(*VmeStartFade*)

| **Command:** | | |
|---|---|---|
| data1-4: | Resource handle from allocation of ResourceVideoCombiner | |
| data5: | Stage (one of enumerated values) | |
| data6 | Transition direction (one of enumerated values) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comment:** | | |
| Enumerated values are described under **VmeStartFade**. | | |

### 80.0A Set Fade Progress

(*VmeSetFadeProgress*)

| **Command:** | | |
|---|---|---|
| data1-4: | Resource handle from allocation of ResourceVideoCombiner | |
| data5: | Stage (one of enumerated values) | |
| data6-9: | Progress (0 through VME_MAX_PROGRESS) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comment:** | | |
| Enumerated values are described under **VmeSetFadeProgress**. | | |

### 80.0B Get Fade Progress

*(VmeGetFadeProgress)*

**Command:**

| | |
|---|---|
| data1-4: | Resource handle from allocation of ResourceVideoCombiner |
| data5: | Stage (one of enumerated values) |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | |
|---|---|
| data1: | Transaction number |
| data2-5: | Progress (0 through VME_MAX_PROGRESS). Negative number indicates error. |

If data2-5 indicates an error, data6-9 are also returned:

| | |
|---|---|
| data6-9: | Error code |

**Comment:**

Enumerated values are described under **VmeGetFadeProgress**.

### 80.0C Set Transition Length

*(VmeSetTransitionLength)*

**Command:**

| | |
|---|---|
| data1-4: | Vme handle |
| data5: | Stage (one of enumerated values) |
| data6-9: | Number of video fields |

**Immediate Reply:**

Ack + transaction number

**Transaction Reply:**

| | | |
|---|---|---|
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |

If data2 indicates an error, data3-6 are also returned:

| | |
|---|---|
| data3-6: | Error code |

**Comment:**

Enumerated values are described under **VmeSetTransitionLength**.

### 80.0D (Unused)

### 80.0E Set Fade Length

*(VmeSetFadeLength)*

| **Command:** | | |
|---|---|---|
| data1-4: | Vme handle | |
| data5: | Stage (one of enumerated values) | |
| data6-9: | Number of video fields | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comment:** | | |
| Enumerated values are described under **VmeSetFadeLength**. | | |

### 80.0F (Unused)

### 80.10 Set Multipliers

*(VmeSetMultipliers)*

| **Command:** | | |
|---|---|---|
| data1-4: | Vme handle | |
| data5: | Stage (one of enumerated values) | |
| data6-9: | Integer multiplier for horizontal (1 to VME_MAX_MULTIPLIER) | |
| data10-13: | Integer multiplier for vertical (1 to VME_MAX_MULTIPLIER) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comment:** | | |
| Enumerated values are described under **VmeSetMultipliers**. | | |

### 80.11 Set Aspect (not fully implemented)

*(VmeSetAspect)*

### 80.12 Set Position

(*VmeSetPosition*)

| **Command:** | |
|---|---|
| data1-4: | Vme handle |
| data5: | Stage (one of enumerated values) |
| data6-9: | X axis screen position (-2* VME_MAX_SCREEN_HEIGHT to +2* VME_MAX_SCREEN_HEIGHT) |
| data10-13: | Y axis screen position (-2* VME_MAX_SCREEN_HEIGHT to +2* VME_MAX_SCREEN_HEIGHT) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |
| **Comment:** | |
| Enumerated values are described under **VmeSetPosition**. | |

### 80.13 Set Rotation

(*VmeSetRotation*)

| **Command:** | |
|---|---|
| data1-4: | Vme handle |
| data5: | Stage (one of enumerated values) |
| data6-9: | Integer representing rotation (0 to VME_MAX_ROTATION) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |
| **Comment:** | |
| Enumerated values are described under **VmeSetRotation**. | |

### 80.14 Set Border Color

(*VmeSetBorderColor*)

| **Command:** | | |
|---|---|---|
| data1-4: | Vme handle | |
| data5: | Stage (one of enumerated values) | |
| data6-9: | Hue, integer representing degrees (0 to VME_MAX_HUE) | |
| data10-13: | Chroma saturation (0 to VME_MAX_CHROMA) | |
| data14-17: | Luminance (0 to VME_MAX_LUMA) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comment:** | | |
| Enumerated values are described under **VmeSetBorderColor**. | | |

### 80.15 Set Border Width

(*VmeSetBorderWidth*)

| **Command:** | | |
|---|---|---|
| data1-4: | Vme handle | |
| data5: | Stage (one of enumerated values) | |
| data6-9: | Width, integer (0 to VME_MAX_BORDER_WIDTH) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comment:** | | |
| Enumerated values are described under **VmeSetBorderWidth**. | | |

### 80.16 Set Edge Softness

*(VmeSetEdgeSoftness)*

| **Command:** | |
|---|---|
| data1-4: | Vme handle |
| data5: | Stage (one of enumerated values) |
| data6-9: | Softness, integer (0 to VME_MAX_SOFTNESS) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |
| **Comment:** | |
| Enumerated values are described under **VmeSetEdgeSoftness**. | |

### 80.17 Set Horizontal Modulation

*(VmeSetHModulation)*

| **Command:** | |
|---|---|
| data1-4: | Vme handle |
| data5: | Stage (one of enumerated values) |
| data6: | Modulation type (one of enumerated values) |
| data7-10: | Amplitude, integer (0 to VME_MAX_MOD_AMP) |
| data11-14: | Frequency, float (0.0 to VME_MAX_MOD_FREQ) |
| data15-18: | Phase, interger (0 to VME_MAX_ROTATION) |
| data19-22: | Delta phase, integer (0 to VME_MAX_ROTATION) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |
| **Comment:** | |
| Enumerated values are described under **VmeSetHModulation**. | |

### 80.18 Set Vertical Modulation

(*VmeSetVModulation*)

| **Command:** | | |
|---|---|---|
| data1-4: | Vme handle | |
| data5: | Stage (one of enumerated values) | |
| data6: | Modulation type (one of enumerated values) | |
| data7-10: | Amplitude, integer (0 to VME_MAX_MOD_AMP) | |
| data11-14: | Frequency, float (0.0 to VME_MAX_MOD_FREQ) | |
| data15-18: | Phase, integer (0 to VME_MAX_ROTATION) | |
| data19-22: | Delta phase, integer (0 to VME_MAX_ROTATION) | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |
| **Comment:** | | |
| Enumerated values are described under **VmeSetVModulation**. | | |

### 80.19 Set Key

*(VmeSetKey)*

| **Command:** | |
|---|---|
| data1-4: | Vme handle |
| data5: | Stage (one of enumerated values) |
| data6-9: | Clip level, integer (0 to VME_MAX_CLIP_LEVEL) |
| data10-13: | Gain, float (VME_MIN_GAIN_VALUE to VME_MAX_GAIN_VALUE) |
| data14: | Invert mode (one of the following values: VmeNormalKey or VmeInvertedKey) |
| data15: | Edge mode (one of the following values: VmeAdditiveKey or VmeMultiplicativeKey) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |
| **Comment:** | |
| Enumerated values are described under **VmeSetKey**. | |

### 80.1A Set Fade Matte Color

*(VmeSetFadeMatteColor)*

| **Command:** | |
|---|---|
| data1-4: | Vme handle |
| data5: | Stage (one of enumerated values) |
| data6-9: | Hue, integer representing degrees (0 to VME_MAX_HUE) |
| data10-13: | Chroma saturation, integer (0 to VME_MAX_CHROMA) |
| data14-17: | Luminance, integer (0 to VME_MAX_LUMA) |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2: | Status of call: |
| | 0: OK |
| | 1: Error condition |
| If data2 indicates an error, data3-6 are also returned: | |
| data3-6: | Error code |
| **Comment:** | |
| Enumerated values are described under **VmeSetFadeMatteColor**. | |

### 80.1B Start Fade At Gen Tc (*VmeStartFadeAtGenTc*)

*Engineering specification not yet available.*

### 80.1C Start Transition At Gen Tc (*VmeStartTransitionAtGenTc*)

*Engineering specification not yet available.*

### 80.1D Get TekVme Major Version Number (*VmeGetMajorVersion*)

**Command:**

no data

**Reply:**

| | |
|---|---|
| data1-4: | Major version number |

### 80.1E Get TekVme Minor Version Number (*VmeGetMinorVersion*)

**Command:**

no data

**Reply:**

| | |
|---|---|
| data1-4: | Minor version number |

### 80.1F Get Transition Type (*VmeGetTransitionType*)

**Command:**

| | |
|---|---|
| data1-4: | Vme handle |
| data5: | Stage (one of enumerated values) |

**Reply:**

| | |
|---|---|
| data1: | Transition type (one of enumerated value) |

**Comment:**

Enumerated values are described under **VmeGetTransitionType**.

### 80.20 Get Transition and Fade Settings (see bitmap)

**Command:**

| | | | |
|---|---|---|---|
| data1-4: | Vme handle | | |
| data5: | Stage (one of enumerated values) | | |
| data6-7: | Bit mask for command | | |
| | bit0: | Transition Length | (*VmeGetTransitionLength*) |
| | bit1: | Reserved | |
| | bit2: | Fade Length | (*VmeGetFadeLength*) |
| | bit3: | Reserved | |

**Reply:**

| | |
|---|---|
| data1-2: | Bit mask for returned values |
| data3-n: | Status values requested (lowest bit first) |

**Comment:**

Enumerated values are described under **VmeGetTransitionLength** and **VmeGetFadeLength**.

### 80.21 Get Position Settings

(see bitmap)

| **Command:** | | | |
|---|---|---|---|
| data1-4: | Vme handle | | |
| data5: | Stage (one of enumerated values) | | |
| data6-7: | Bit mask for command | | |
| | bit0: | Multipliers | (*VmeGetMultipliers*) |
| | bit2: | Position | (*VmeGetPosition*) |
| | bit3: | Rotation | (*VmeGetRotation*) |
| **Reply:** | | | |
| data1-2: | Bit mask for returned values | | |
| data3-n: | Status values requested (lowest bit first) | | |
| Multipliers and Position return two 4-byte integers | | | |
| Rotation returns a single 4-byte integer | | | |
| **Comment:** | | | |
| Enumerated values are described under **VmeGetMultipliers**, **VmeGetPosition**, and **VmeGetRotation**. | | | |

### 80.22 Get Border and Edge Settings

(see bitmap)

| **Command:** | | | |
|---|---|---|---|
| data1-4: | Vme handle | | |
| data5: | Stage (one of enumerated values) | | |
| data6-7: | Bit mask for command | | |
| | bit0: | Border color | (*VmeGetBorderColor*) |
| | bit1: | Border width | (*VmeGetBorderWidth*) |
| | bit2: | Edge softness | (*VmeGetEdgeSoftness*) |
| **Reply:** | | | |
| data1-2: | Bit mask for returned values | | |
| data3-n: | Status values requested (lowest bit first) | | |
| Border color returns three integers (hue, chroma, and luminance) | | | |
| Border width and edge softness return a single 4-byte integer | | | |
| **Comment:** | | | |
| Enumerated values are described under **VmeGetBorderColor**, **VmeGetBorderWidth**, and **VmeGetEdgeSoftness**. | | | |

### 80.23 Get Horizontal Modulation Settings

*(VmeGetHModulation)*

| **Command:** | | |
|---|---|---|
| data1-4: | Vme handle | |
| data5: | Stage (one of enumerated values) | |
| **Reply:** | | |
| data1: | Success status: | |
| | 0: | OK |
| | 1: | Command failure |
| data2: | Modulation type (one of enumerated values) | |
| data3-6: | Amplitude (integer) | |
| data7-10: | Frequency (float) | |
| data11-14: | Phase (integer) | |
| data15-18: | Delta phase (integer) | |
| **Comment:** | | |

Enumerated values are described under **VmeGetHModulation**.

### 80.24 Get Vertical Modulation Settings

*(VmeGetVModulation)*

| **Command:** | | |
|---|---|---|
| data1-4: | Vme handle | |
| data5: | Stage (one of enumerated values) | |
| **Reply:** | | |
| data1: | Success status: | |
| | 0: | OK |
| | 1: | Command failure |
| data2: | Modulation type (one of enumerated values) | |
| data3-6: | Amplitude (integer) | |
| data7-10: | Frequency (float) | |
| data11-14: | Phase (integer) | |
| data15-18: | Delta phase (integer) | |
| **Comment:** | | |

Enumerated values are described under **VmeGetVModulation**.

### 80.25 Get Key Settings

(*VmeGetKey*)

| **Command:** | | |
|---|---|---|
| data1-4: | Vme handle | |
| data5: | Stage (one of enumerated values) | |
| **Reply:** | | |
| data1: | Success status: | |
| | 0: | OK |
| | 1: | Command failure |
| data2-5: | Clip level (integer) | |
| data6-9: | Gain (float) | |
| data10: | Invert mode (one of enumerated values) | |
| data11: | Edge mode (one of enumerated values) | |
| If data1 indicates a command failure, data2-5 are returned: | | |
| data2-5: | Error code | |
| **Comment:** | | |
| Enumerated values are described under **VmeGetKey**. | | |

### 80.26 Get Fade Matte Color Settings

(*VmeGetFadeMatteColor*)

| **Command:** | | |
|---|---|---|
| data1-4: | Vme handle | |
| data5: | Stage (one of enumerated values) | |
| **Reply:** | | |
| data1: | Success status: | |
| | 0: | OK |
| | 1: | Command failure |
| data2-5: | Hue (integer) | |
| data6-9: | Chroma (integer) | |
| data10-13: | Luminance (integer) | |
| If data1 indicates a command failure, data2-5 are returned: | | |
| data2-5: | Error code | |
| **Comment:** | | |
| Enumerated values are described under **VmeGetFadeMatteColor**. | | |

# Streaming transfer access commands (0x90)

### 90.00 Xfr Request

*(XfrRequest)*

| **Command:** | |
|---|---|
| data1-n: | Null-terminated UML string for source |
| data(n+1) - (n+2): | One-byte destination count |
| data(n+3) - m: | Null-terminated UML string for destination |
| **Immediate Reply:** | |
| Ack + transaction number | |
| **Transaction Reply:** | |
| data1: | Transaction number |
| data2-5: | XfrToken |
| If data2-5 are null, data6-9 are also returned: | |
| data6-9: | Error code |

### 90.01 Xfr Get Status

(*XfrGetStatus*)

| **Command:** | | |
|---|---|---|
| data1-4: | XfrToken | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| if data2 = OK, data3-54 will contain a series of four-byte status values. (See *xfrtypes.h* for more info.) | | |
| data3-6: | State | |
| data7-10: | Error | |
| data11-14: | Percent completed | |
| data15-18: | Completed bytes (high-order bytes) | |
| data19-22: | Completed bytes (low-order bytes) | |
| data23-26: | Total bytes (high-order bytes) | |
| data27-30: | Total bytes (low-order bytes) | |
| data31-34: | Completed fields (high-order bytes) | |
| data35-38: | Completed fields (low-order bytes) | |
| data39-42: | Total fields (high-order bytes) | |
| data43-46: | Total fields (low-order bytes) | |
| data47-50: | Average bandwidth | |
| data51-54: | Last bandwidth | |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 90.02 Xfr Abort

(*XfrAbort*)

| **Command:** | | |
|---|---|---|
| data1-4: | XfrToken | |
| **Immediate Reply:** | | |
| Ack + transaction number | | |
| **Transaction Reply:** | | |
| data1: | Transaction number | |
| data2: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data2 indicates an error, data3-6 are also returned: | | |
| data3-6: | Error code | |

### 90.03 Xfr Get Active Tokens

*(XfrGetActiveTokens)*

| **Command:** | | |
|---|---|---|
| data1-4: | Maximum entries to return (max of 100) | |
| **Reply:** | | |
| data1: | Status of call: | |
| | 0: | OK |
| | 1: | Error condition |
| If data1 = OK, the following values are also returned: | | |
| data2-5: | Return count | |
| data6-n: | Series of 4-byte XfrTokens | |
| If data1 indicates an error, data2-5 are also returned: | | |
| data2-5: | Error code | |

Appendix **A**

# Odetics, VDCP, and BVW Protocols

This appendix provides information on supported and unsupported third-party protocols by Odetics, VDCP, and BVW.

Recent additions to our third-party protocol support include DVCPRO support (Odetics, VDCP, BVW, and BVW-insert Edit protocols) and VDCP control support for Fibre Channel streaming (copying and multicasting).

## Odetics commands

This section lists the Odetics Broadcast video disk recorder commands. The Profile server supports some, but not all, Odetics commands. (Supported commands are listed in *Table 25*; unsupported commands are listed in *Table 26*.) Each command is identified by its command code and name. For more information, refer to the *Odetics Broadcast Video Disk Recorder Command and Control Specification manual (part number 4136201s-C*).

Most of the commands and responses are identical to those specified in the Sony Betacam SP Protocol. Information listed here is intended to specify the minimum requirements for each command. Unless otherwise specified, the Profile server implements all the capabilities for each command as specified.

Some commands have been modified from the specifications in the Sony Betacam/Betacam SP Protocol. These modifications were developed to address capabilities required for the video disk recorders, but are not available in current video cassette recorders. These modifications follow the original command format and function as closely as possible.

Some commands have been added to the command set. These commands have been developed to address video disk recorder capabilities not available in current cassette technology. These commands were created only when it became apparent that the current command set was not applicable.

**Table 25. Supported Odetics commands**

| Command | Name | Comments |
|---|---|---|
| **System control** | | |
| 12.11 | Device Type | Returns BVW75 |
| **System control returns** | | |
| 10.01 | ACK | |
| 11.12 | NAK | |
| 00.11 | Device Type Request | |
| **Transport controls** | | |
| 20.00 | Stop | |
| 20.01 | Play | |
| 20.02 | Record | |
| 20.04 | Standby Off | No effect other than status change (responds with Ack). |
| 20.05 | Standby On | No effect other than status change (responds with Ack). |
| 20.0F | Eject | No effect other than status change (responds with Ack). |
| 20.10 | Fast Forward | |
| 2X.11 | Jog Forward | |
| 2X.12 | Variable Forward | |
| 2X.13 | Shuttle Forward | |
| 20.20 | Rewind | |
| 2X.21 | Jog Reverse | |
| 2X.22 | Variable Reverse | |
| 2X.23 | Shuttle Reverse | |
| 2X.31 | Cue up w/data | Odetics recommends that this command no longer be used. Use InPreset in AutoMode instead. |
| 20.52 | Tension Release | No effect other than status change (responds with Ack). |
| **Preset controls** | | |
| 4X.14 | In Preset | See *Odetics protocol notes* on page 773. |
| 4X.15 | Out Preset | |
| 40.20 | In Reset | |
| 40.21 | Out Reset | |
| 40.40 | Auto Mode Off | |
| 40.41 | Auto Mode On | |
| **Sense requests** | | |
| 61.0C | Current Time Sense | Use 04 for data-1 to request the timer-1 value; this is the value referred to by arguments to CueUpWithData, InPreset, OutPreset, PreviewInPreset, and PreviewOutPreset. |
| 61.20 | Status Sense | |
| 60.36 | Time Mode Sense | Always replies Timer1, which is the mode used for Odetics cueing. |
| **ID controls** | | |
| A0.01 | Auto Skip | |

**Table 25. Supported Odetics commands (Continued)**

| Command | Name | Comments |
|---|---|---|
| AX.02 | Record Cue up w/data | See *Odetics protocol notes* on page 773. |
| AX.04 | Preview In preset | See *Odetics protocol notes* on page 773. |
| AX.05 | Preview Out Preset | |
| A0.06 | Preview In reset | |
| A0.07 | Preset Out reset | |
| A8.10 | Erase ID | |
| A8.11 | Erase segment | |
| A0.14 | List first ID | |
| A0.15 | List next ID | |
| A8.18 | ID status request | |
| A0.1C | Storage request | |
| A8.20 | Set device ID | |
| A0.21 | Device ID request | |

**Table 26. Unsupported Odetics commands**

| Command | Name | Comments |
|---|---|---|
| **Preset controls** | | |
| 44.05 | User Bits Preset | |
| **ID controls** | | |
| A0.22 | Hard error request | Odetics has removed this command from the protocol. |
| AX.23 | Set compression rate | Odetics has removed this command from the protocol. |

## Odetics protocol notes

1. The default ID code is set to a device-specific name (for PDR100, “OdeticsDefault”).
2. Sending an ID code in any of these commands...

   ```
   CueUpWithData (28.31, 2C.31)
   InPreset (48.14, 4C.14)
   RecordCueUpWithData (A8.02, AC.02)
   PreviewInPreset (A8.04, AC.04)
   ```

   ...will set the default ID to the ID used in the command.
3. When the following commands are sent...

   ```
   CueUpWithData (20.31, 24.31)
   InPreset (40.14, 44.14)
   RecordCueUpWithData (A0.02, A4.02)
   ```

   ...the active ID will be...

   ```
   if (there are >= 2 clips already on the timeline {
   the ID of the second-to-last clip on the timeline
   }
   else if (there is 1 clip already on the timeline {
   the ID of the clip on the timeline
   ```

```
}
else {
the default ID
}
```

4. When the following command is sent...

```
PreviewInPreset (A8.04, AC.04)
if (there are >= 2 clips already on the timeline {
the ID of the last clip on the timeline
}
else if (there is 1 clip already on the timeline {
the ID of the clip on the timeline
}
else {
the default ID
}
```

...loading two groups of clips is disabled.

# VDCP commands

This section lists the VDCP video disk communication protocol commands. The Profile server supports some, but not all, VDCP commands. (Supported commands are listed in *Table 27*; unsupported commands are listed in *Table 28*.) Each command is identified by its command code and name. For more information, refer to the *Video Disk Communications Protocol* manual.

VDCP control has been enhanced to utilize Fibre Channel streaming for copies and multicast.

**Table 27. Supported VDCP commands**

| Command | Name | Comments |
|---|---|---|
| **System controls** | | |
| 0X.11 | Device Type Request | |
| 0X.14 | Delete From Archive | |
| 0X.15 | Protect ID | |
| 0X.16 | Unprotect ID | |
| **Immediate controls** | | |
| 1X.00 | Stop | |
| 1X.01 | Play | |
| 1X.02 | Record | |
| 1X.04 | Still | |
| 1X.05 | Step | |
| 1X.06 | Continue | |
| 1X.07 | Jog | |
| 1X.08 | Vari. Play | |
| 1X.0A | EE Mode | |
| **Preset controls** | | |
| 2X.1D | Rename ID | First eight characters are existing name of ID, second eight characters are new name of ID. This sets the CLIPS_ADDED and CLIPS_DELETED bits. |
| 2X.1F | New Copy | Supported only in VdrPanel V4.09 and higher. |
| 2X.20 | Sort Mode | |
| 2X.21 | Close Port | |
| 2X.22 | Select Port | |
| 2X.23 | Record Init | |
| 2X.24 | Play Cue | |
| 2X.25 | Play Cue with Data | |
| 2X.26 | Delete | |
| 2X.27 | Get From Archive | |
| 2X.2A | Send To Archive | |
| 2X.2B | % to Signal Full | |
| 2X.2C | Record Init with Data | |

**Table 27. Supported VDCP commands (Continued)**

| Command | Name | Comments |
|---|---|---|
| 2X.31 | Video Compression Rate | The least significant byte of the rate is transmitted first and the most significant byte last. |
| 2X.37 | Video Compression Parameters | Data transmitted is (in order of transmission) minimum luminance Q, maximum luminance Q, minimum chrominance Q, and maximum chrominance Q. For each of the four parameters, the transmitted value is an IEEE single-precision floating-point number with the least significant byte transmitted first and the most significant byte transmitted last (for a total of 16 data bytes). |
| 2X.3A | Record Mode | |
| 2X.43 | Disk Preroll | Supported only in VdrPanel V3.33 and higher. |
| 2X.50 | Copy File To | |
| 2X.51 | Delete File From | |
| 2X.52 | Abort Copy File To | |
| Sense requests | | |
| 3X.01 | Open Video Port | |
| 3X.02 | Next | |
| 3X.05 | Port Status Request | Video type bits (Status 5) are not supported. |
| 3X.06 | Position Request | Time type “2” (timecode) is not supported but will return the position (same as time type “1”). |
| 3X.07 | Active ID Request | |
| 3X.08 | Device Type Request | |
| 3X.10 | System Status Request | |
| 3X.11 | List | |
| 3X.14 | ID Size Request | The duration returned is from the first recorded frame to the last recorded frame (inclusive). If the first recorded frame is not at position 00:00:00:00, a second value--the time of the first recorded frame--is returned. This second value can be used with **Play Cue With Data** (2x.25) to start playback at the first recorded frame. |
| 3X.15 | IDs Added To Archive | |
| 3X.16 | ID Request | Supports DELETE_PROTECTED bit.<br>Video type bits are not supported. |
| 3X.17 | Compression Settings Request | Dummy handler |
| 3X.18 | IDs Added | |
| 3X.19 | IDs Deleted | |
| **Ack and Nak do not use header (no STX or byte count)** | | |
| 04 | ACK | |
| 05 | NAK | |

**Table 28. Unsupported VDCP commands**

| Command | Name | Comments |
|---|---|---|
| **System controls** | | |
| 0X.0C | Local Disable | |
| 0X.0D | Local Enable | |
| **Immediate controls** | | |
| 1X.03 | Freeze | |
| 1X.09 | Unfreeze | |
| **Preset controls** | | |
| 2X.1E | Preset Standard | |
| 2X.29 | Clear | |
| 2X.2D | Select Logical Drive | |
| 2X.2E | System Delete ID | |
| 2X.30 | Preset | |
| 2X.32 | Audio Sample Rate | |
| 2X.33 | Audio Compression Rate | |
| 2X.34 | Audio IN Level | Syntax undefined |
| 2X.35 | Audio OUT Level | Syntax undefined |
| 2X.38 | Select Output | |
| 2X.39 | Select Input | |
| 2X.41 | Subcarrier Adjust | |
| 2X.42 | Horizontal Sync Timing | Syntax undefined |
| **Sense requests** | | |
| 3X.03 | Last | |
| 3X.25 | Multi Port Status Request | |

## VDCP states

This section documents a number of states and state transitions related to the port status 1 bits listed in *Table 29* below.

**Table 29. Port status 1 bits**

| Bit | Description |
|---|---|
| CD | cue/init done (byte 0, bit 7) |
| V | variable play (byte 0, bit 5) |
| J | jog (byte 0, bit 4) |
| S | still (byte 0, bit 3) |
| P | play or record (byte 0, bit 2) |
| CI | cue/init (byte 0, bit 1) |
| I | idle (byte 0, bit 0) |

The jog bit (J) is asserted only during the extremely short time that is required to move between video fields and is cleared when the channel is displaying the target video field. Additionally, the jog bit is asserted during execution of a step command and cleared at the end of execution of the step command.

The **State number** column indicates the number of the state assigned to a particular combination of these status bits.

The **Command (action)** column includes RS-422 commands received in that state or asynchronous changes in the listener (enclosed in parentheses).

The **Port status 3 error bits** column indicates any port status 3 error bits asserted after receiving the given commands in that state. See *Table 31, Port status 3 error bits* on page 781 for an explanation of the abbreviations.

The **Notes** column includes information about what should be done in the transition to the next state, as described in *Table 32, Transition from state to state* on page 781.

The **Next State** column indicates the next state after the given command or action occurs, along with a (redundant) listing of the port status 1 bits for the next state.

**Table 30. VDCP states**

| State number | Port status 1 bits | Command (action) | Port status 3 error bits | Notes | Next state |
|---|---|---|---|---|---|
| 0 | I | 1x.00 Stop | | | 0 (I) |
| | | 1x.01 Play 1x.02 Record | PNA | | 0 (I) |
| | | 1x.04 Still 1x.05 Step 1x.06 Continue 1x.07 Jog 1x.08 VariablePlay | PNA | | 0 (I) |
| | | 2x.24 PlayCue 2x.25 CueWithData 2x.23 RecordInit 2x.2C RecordInitWithData | | | 1 (CI) |
| 1 | CI | 1x.00 Stop | | EC | 0 (I) |
| | | 1x.01 Play 1x.02 Record | CND | | 1 (CI) |
| | | 1x.04 Still 1x.05 Step 1x.06 Continue 1x.07 Jog 1x.08 VariablePlay | CND | | 1 (CI) |
| | | 2x.24 PlayCue 2x.25 CueWithData 2x.23 RecordInit 2x.2C RecordInitWithData | | CNC | 1 (CI) |
| | | (Cue completes) | | | 2 (CD) |
| 2 | CD | 1x.00 Stop | | EC | 0 (I) |
| | | 1x.01 Play 1x.02 Record | | | 3 (P) |
| | | 1x.04 Still 1x.05 Step 1x.07 Jog | | | 6 (S P) |
| | | 1x.06 Continue | | | 3 |
| | | 1x.08 VariablePlay | | | 7 (V P) |
| | | 2x.24 PlayCue 2x.25 CueWithData 2x.23 RecordInit 2x.2C RecordInitWithData | | CNC | 1 (CI) |
| 3 | P | 1x.00 Stop | | EC | 0 (I) |
| | | 1x.01 Play 1x.02 Record | PNA | | 3 (P) |
| | | 1x.04 Still 1x.05 Step 1x.07 Jog | | | 6 (S P) |
| | | 2x.24 PlayCue 2x.25 CueWithData 2x.23 RecordInit 2x.2C RecordInitWithData | | | 4 (P CI) |
| 4 | P CI | 1x.00 Stop | | EC | 0 (I) |
| | | 1x.01 Play 1x.02 Record | CND | | 4 (P CI) |
| | | 1x.04 Still 1x.05 Step 1x.07 Jog | | | 10 (S P CI) |
| | | 1x.08 VariablePlay | | | 11 (V P CI) |
| | | (Cue completes) | | | 5 (P CD) |
| 5 | P CD | 1x.01 Play 1x.02 Record | | PCC | 3 (P) |
| | | 1x.04 Still 1x.05 Step 1x.07 Jog | | | 8 (S P CD) |
| | | 1x.06 Continue | | CP | 5 (P CD) |
| | | 1x.08 VariablePlay | | | 9 (V P CD) |
| | | 2x.24 PlayCue 2x.25 CueWithData 2x.23 RecordInit 2x.2C RecordInitWithData | | CNC | 4 (P CI) |
| | | (Play transitions to cued clip) | | | 2 (CD) |

**Table 30. VDCP states (Continued)**

| State number | Port status 1 bits | Command (action) | Port status 3 error bits | Notes | Next state |
|---|---|---|---|---|---|
| 6 | S P | 1x.00 Stop | | EC | 0 (I) |
| | | 1x.01 Play 1x.02 Record | PNA | | 6 (S P) |
| | | 1x.04 Still 1x.05 Step 1x.07 Jog | | | 6 (S P) |
| | | 1x.06 Continue | | CP | 3 (P) |
| | | 1x.08 VariablePlay | | | 7 (V P) |
| | | 2x.24 PlayCue 2x.25 CueWithData 2x.23 RecordInit 2x.2C RecordInitWithData | | | 10 (S P CI) |
| 7 | V P | 1x.00 Stop | | EC | 0 (I) |
| | | 1x.01 Play 1x.02 Record | PNA | | 7 (V P) |
| | | 1x.04 Still 1x.05 Step 1x.07 Jog | | | 6 (S P) |
| | | 1x.06 Continue | | CP | 3 (P) |
| | | 1x.08 VariablePlay | | | 7 (V P) |
| | | 2x.24 PlayCue 2x.25 CueWithData 2x.23 RecordInit 2x.2C RecordInitWithData | | | 11 (V P CI) |
| 8 | S P CD | 1x.01 Play 1x.02 Record | | PCC | 3 (P) |
| | | 1x.04 Still 1x.05 Step 1x.07 Jog | | | 8 (S P CD) |
| | | 1x.06 Continue | | | 5 (P CD) |
| | | 1x.08 VariablePlay | | | 9 (V P CD) |
| | | 2x.24 PlayCue 2x.25 CueWithData 2x.23 RecordInit 2x.2C RecordInitWithData | | CNC | 10 (S P CI) |
| 9 | V P CD | 1x.01 Play 1x.02 Record | | PCC | 3 (P) |
| | | 1x.04 Still 1x.05 Step 1x.07 Jog | | | 8 (S P CD) |
| | | 1x.06 Continue | | | 5 (P CD) |
| | | 1x.08 VariablePlay | | | 9 (V P CD) |
| | | 2x.24 PlayCue 2x.25 CueWithData 2x.23 RecordInit 2x.2C RecordInitWithData | | | 11 (V P CI) |
| | | (Play transitions to cued clip) | | | ? |
| 10 | S P CI | 1x.00 Stop | | EC | 0 (I) |
| | | 1x.01 Play 1x.02 Record | CND | | 10 (S P CI) |
| | | 1x.04 Still 1x.05 Step 1x.07 Jog | | | 10 (S P CI) |
| | | 1x.08 VariablePlay | | | 11 (V P CI) |
| | | 2x.24 PlayCue 2x.25 CueWithData 2x.23 RecordInit 2x.2C RecordInitWithData | | CNC | 10 (S P CI) |
| | | (Cue completes) | | | 8 (S P CD) |

**Table 30. VDCP states (Continued)**

| State number | Port status 1 bits | Command (action) | Port status 3 error bits | Notes | Next state |
|---|---|---|---|---|---|
| 11 | V P CI | 1x.00 Stop | | EC | 0 (I) |
| | | 1x.01 Play 1x.02 Record | CND | | 11 (V P CI) |
| | | 1x.04 Still 1x.05 Step 1x.07 Jog | | | 10 (S P CI) |
| | | 1x.08 VariablePlay | | | 11 (V P CI) |
| | | 2x.24 PlayCue 2x.25 CueWithData 2x.23 RecordInit 2x.2C RecordInitWithData | | CNC | 11 (V P CI) |
| | | (Cue completes) | | | 9 (V P CD) |
| | | (Play transitions to cued clip) | | | ? |

**Table 31. Port status 3 error bits**

| Error bits | Description |
|---|---|
| NCI | Not in Cue/Init state (byte 3, bit 0) |
| PNA | Port Not Active (byte 3, bit 4) |
| CND | Cue Not Done (byte 3, bit 1) |

**Table 32. Transition from state to state**

| Transition | Description |
|---|---|
| PCC | Start play on cued clip |
| CP | Continue play on currently playing clip |
| CNC | Detach cued or cueing clip and cue new clip |
| EC | Eject clip(s) |

# BVW commands and responses

This section lists the BVW protocol commands. The Profile server supports the commands listed in *Table 33, Supported BVW commands.* For more information, refer to the second edition or later of the *Sony Videocassette Recorder Bvw Series Protocol Manual.*

The commands listed here are either implemented or stubbed (returning only a command acknowledgment) as indicated. Each command is identified by its command code and name.

Information listed here is intended to specify the minimum requirements for each command. Unless otherwise specified, the Profile server implements all the capabilities for each command as specified in the Sony documentation. For more information, refer to the second edition or later of the *Sony Videocassette Recorder Bvw Series Protocol Manual* (part number 9-977-544-11).

## Command list

The following table lists the VTR protocol commands. The Profile server returns an ACK response to each command, unless otherwise indicated.

**Table 33. Supported BVW commands**

| Command | Name | Return |
|---|---|---|
| 0X.11 | Device Type Request | The controller can distinguish Profile from BVW-75:<br>If 00.11 is sent, returns 20.25 or 21.25 (for 525- or 625- line operation respectively).<br>If 01.11.00 is sent, returns D8.00 (disk recorder).<br>This allows the controller to decide whether to cue-up by a combination of FastForward/Rewind and CueUpWithData (faster with BVW-75) or CueUpWithData (faster with Profile). |
| 20.00 | Stop | |
| 20.01 | Play | |
| 20.02 | Record | |
| 20.04 | Standby Off | Stubbed. Acknowledges command only. |
| 20.05 | Standby On | Stubbed. Acknowledges command only. |
| 20.0F | Eject | |
| 20.10 | Fast Forward | |
| 2X.11 | Jog Forward | |
| 2X.12 | Variable Forward | |
| 2X.13 | Shuttle Forward | |
| 20.20 | Rewind | |
| 2X.21 | Jog Reverse | |
| 2X.22 | Variable Reverse | |
| 2X.23 | Shuttle Reverse | |
| 20.30 | Preroll | |

**Table 33. Supported BVW commands (Continued)**

| Command | Name | Return |
|---|---|---|
| 24.31 | Cue Up With Data | Matches behavior of BVW-75 as follows:<br>After start-up, or after the displayed timecode is changed from the "Options/Select Timecode ..."dialog box, CueUpWithData cues up using the following time codes:<br>VITC if VITC recorder is displayed<br>LTC if LTC recorder is displayed<br>Timer1 otherwise<br>After TimerModeSelect (41.36) has been sent, CueUpWithData cues up using the following time codes:<br>VITC if 41.36.00 is sent and VITC is displayed<br>LTC if 41.36.00 is sent and anything else is displayed<br>Timer1 if 41.36.01 is sent |
| 21.38 | Program Play + | |
| 21.39 | Program Play - | |
| 20.52 | Tension Release | Stubbed. Acknowledges command only. |
| 20.60 | Full EE Off | Panel indicator (PB/EE) changes to match. |
| 20.61 | Full EE On | Panel indicator changes to EE only. This causes timecode, video, and audio to go to EE in the BVW (insert-edit) controller, and video and audio to go to EE in the BVW controller. |
| 20.63 | Selected EE On | |
| 20.64 | Edit Off | |
| 20.65 | Edit On | |
| 44.00 | Timer-1 Preset | |
| 44.04 | Time Code Preset | |
| 44.05 | User Bit Reset | |
| 44.08 | Timer-1 Reset | |
| 40.10 | In Entry | |
| 40.11 | Out Entry | |
| 44.14 | In Preset | |
| 44.15 | Out Preset | |
| 40.20 | In Reset | |
| 40.21 | Out Reset | |
| 40.24 | In Recall | |
| 40.25 | Out Recall | |
| 4X.30 | Edit Preset | |
| 44.31 | Preroll Time Preset | |
| 41.34 | Head Select | Stubbed. Acknowledges command only. |
| 41.36 | Timer Mode Select | |
| 40.40 | Auto Mode Off | Stubbed. Acknowledges command only. |
| 40.41 | Auto Mode On | Stubbed. Acknowledges command only. |
| 61.0C | Current Time Sense | |
| 60.10 | In Data Sense | |
| 60.11 | Out Data Sense | |

**Table 33. Supported BVW commands (Continued)**

| Command | Name | Return |
|---|---|---|
| 61.20 | Status Sense | |
| 6X.30 | Edit Preset Sense | |
| 60.31 | Preroll Time Sense | |
| 60.36 | Timer Mode Sense | |
| A0:01 | Enable Clip Loading | |
| A0:02 | Disable Clip Loading | |
| A0:10 | Clip Length Request | Clip length response in the form 00:00:00:00. |
| A8:12 | Set Clip Length | |
| A0:20 | Find First Clip Request | Clip name. |
| A0:21 | Find Next Clip Request | Clip name. |
| AX:30 | Load Clip | |
| AX:40 | New Clip | |
| AX:50 | Delete Clip | |

# Index


## A

Activate Resource 731
Allocate Context 750
Allocate Resource 726
API libraries 7
    TekCfg library 9, 64
    TekPdr library 10, 85
    TekPls library 14, 195
    TekRem library 17, 272
    TekVdr library 18, 281
    TekVfs library 24, 471
    TekVme library 26, 514
    TekXfr library 28, 559
Apply Context 750
Attach Media 711
Attach Media By Movie With Marks And Fades 718
Attach Media with Marks 712
Attach Media With Marks And Fades 717
Attach Movie 684
Attach Movie With Marks 685, 694
Attach Movie With Resources 690
Attach Open Movie 692
Attach Open Movie By Port 697
Attach Open Movie With Marks 695
Attach Open Movie With Marks By Port 698

## B

black media 92, 129, 194
BVW commands and responses 782

## C

Cancel At Gen Tc Cmds 657
Cancel Event 745
CfgGetCodecAttributes 64, 579
CfgGetCodecTimingMode 65, 592
CfgGetFileSystemName 66, 575
CfgGetInputAttributes 67, 579
CfgGetInputName 68, 575
CfgGetInputUnitNum 69, 576
CfgGetMajorVersion 70, 578
CfgGetMinorVersion 71, 579
CfgGetNumCodecs 72, 575
CfgGetNumFileSystems 73, 574
CfgGetNumInputs 74, 575
CfgGetNumOutputs 75, 575
CfgGetNumResources 76, 593
CfgGetNumTCGenerators 77, 575
CfgGetNumTCRecorders 78, 575
CfgGetOutputAttributes 79, 579
CfgGetOutputName 80, 576
CfgGetOutputUnitNum 81, 576
CfgGetResourceName 82, 593
CfgGetStandard 83, 576
CfgGetVideoInputStandard 84, 576
Clear Timeline 737
Close Port 649
Common Movie Format 85
Copy Media Token 703
Create Black Media Token 702
Create Directory 581
Create Extension 597
Create Media Token 702
Create Media Token with Path 703
Create Track Token 705
crosspoint lines 571
Cue Play 650
Cue Record 652

## D

Deactivate Resource 732
Default Event 743
Default Event At Gen Tc 747
Delete Extension 645
Delete Media 703
Detach Media 713
Detach Movie 686
Directory Exists 581
Disable Movie Position Limits 692
DVCPRO 284, 771

## E

Enable Movie Position Limits 693
Event Poll 594
event types

EventConnectResources 305, 414
EventMixAudio 305, 414
EventSetGtcBits 305, 414
EventSetGtcTime 305, 414
EventVoiceOver 306, 415
Expand Movie 691
Expand Movie By Port 696
Extension Command Execution 646

## F

Fibre Channel
UML usage 562
Free Context 750

## G

Get Abs Port Config 739
Get Attached Media Status 710
Get Attached Movie Status 682
Get Attributes by Type 579
Get Border and Edge Settings 764
Get Current Directory 577
Get Current Drive 577
Get Default Codec Timing 592
Get Device ID 574
Get Disk Capacity 574
Get Engine Major Version Number 578
Get Engine Minor Version Number 578
Get Fade Matte Color Settings 766
Get Fade Progress 755
Get File System Name 575
Get First Timecode Position 676
Get Horizontal Modulation Settings 765
Get Input Name 575
Get Input Unit 576
Get Key Settings 766
Get Last Timecode Position 676
Get Maximum Number of Tracks 700
Get Media at Position 714
Get Media Audio Fades 717
Get Media State 707
Get Media State Info 708
Get Media Token at Position 704
Get Movie at Position 687
Get Movie at Resource Position 691
Get Mpeg Gop Limits 740
Get Next Event Handle 746
Get Next Media 713
Get Next Media Token 703
Get Next Movie 686
Get Next Open Port 656
Get Next Resource 730
Get Next Track 701
Get Null Media Token 706
Get Number of File Systems 574
Get Number of Media on Track 702
Get Number of Resources 593
Get Number of Tracks 701
Get Number of Types 575
Get Output Name 576
Get Output Unit 576
Get Port Handle from Event Handle 746
Get Port Handle From Media Handle 716
Get Port Handle from Movie Handle 689
Get Port Handle from Resource Handle 737
Get Port Status 647
Get Position Limits 656
Get Position Settings 764
Get Previous Event Handle 746
Get Previous Media 713
Get Previous Media Token 704
Get Previous Movie 687
Get Previous Open Port 656
Get Previous Resource 730
Get Previous Track 701
Get Protocol Version 573
Get Resource Configuration 723
Get Resource Connect Handle 728
Get Resource From Media 717
Get Resource Name 593
Get Resource Status 721
Get Resource Type From Track Type 706
Get Resource User Data 738
Get Standard 576
Get Stored Media Status 699
Get TekCfg Major Version Number 578
Get TekCfg Minor Version Number 579
Get TekPdr Major Version Number 580
Get TekPdr Minor Version Number 580
Get TekVdr Major Version Number 577
Get TekVdr Minor Version Number 577

Get TekVfs Major Version Number 580
Get TekVfs Minor Version Number 581
Get TekVme Major Version Number 763
Get TekVme Minor Version Number 763
Get Track Length 701
Get Track Token From Media Token 702
Get Track Token Number 705
Get Track Token Type 706
Get Track Type From Resource 706
Get Transaction Reply 565, 573
Get Transition and Fade Settings 763
Get Transition Progress 753
Get Transition Type 763
Get Vertical Modulation Settings 765
Get Video Combiner Part Handle 738
Get Video Field Count 580
Get Video Format 578
Get Video Standard 576

## H

header files
    pls_sp.h 595
    plsconst.h 595

## I

Idle 650
Intel GNU general public license agreement 4
Is Media Token Null 706
Is Movie Position Limiting Enabled 693
Is Resource Available 737

## J

Jog Play 651
Jog Record 653
JPEG software license agreement 3

## L

LOCALHOST 562
Louth commands 775

## M

Merge Resources 739
mix effects board 514
Movie To Media Name 677

## N

Named Configuration Space 569, 649

## O

Odetics commands 771
Open Port 649

## P

Pdr Backup Movie Descriptor 672
Pdr Close All Enumerators 680
Pdr Close Find 660
Pdr Close Movie 664
Pdr Copy Movie 668
Pdr Copy Movie Token 664
Pdr Create Event 674
Pdr Create Movie 663
Pdr Delete Movie 665
Pdr Find First Dataset 660
Pdr Find First Group 661
Pdr Find First Movie 663
Pdr Find Next Dataset 660
Pdr Find Next Group 662
Pdr Find Next Movie 663
Pdr Get Current Dataset 661
Pdr Get Current Group 662
Pdr Get List of Names 673
Pdr Get Movie Changes 674
Pdr Get Movie State 666
Pdr Get Movie State Info 681
Pdr Get Movie Status 659
Pdr Get Number of Movies 667
Pdr Get Registry 675
Pdr Get Small User Data 678
Pdr Get User Data 671
Pdr Get User Data Commit 672
Pdr Get Wait Operation Status 669
Pdr Movie Exists 667
Pdr Open Movie 664
Pdr Rename Movie 665
Pdr Restore Movie Descriptor 673
Pdr Save Movie 669
Pdr Save Movie By Port 679
Pdr Set Clip Marks 667
Pdr Set Current Dataset 661

Pdr Set Current Group 662
Pdr Set Movie Attributes 666
Pdr Set Registry 675
Pdr Set Small User Data 678
Pdr Set User Data 670
Pdr Set User Data Commit 671
Pdr Terminate Wait Operation 670
Pdr Trim Clip 668
PdrBackupMovieDescription 85, 672
PdrCloseAllEnumTokens 86, 680
PdrCloseFind 87, 660
PdrCloseMovie 88, 664
PdrCopyMediaToken 89, 703
PdrCopyMovie 90, 668
PdrCopyMovieToken 91, 664
PdrCreateBlackMediaToken 92, 702
PdrCreateEvent 93, 594, 674
PdrCreateMediaToken 94, 702
PdrCreateMediaTokenWithPath 95, 703
PdrCreateMovie 96, 663
PdrCreateTrackToken 97, 705
PdrDeleteExtensionAtPos 98
PdrDeleteMedia 99, 703
PdrDeleteMovie 100, 665
PdrFindFirstDataset 101, 660
PdrFindFirstExtensionPos 102
PdrFindFirstGroup 103, 661
PdrFindFirstMediaUsage 104
PdrFindFirstMovie 105, 663
PdrFindNextDataset 106, 660
PdrFindNextExternsionPos 107
PdrFindNextGroup 108, 662
PdrFindNextMediaUsage 109
PdrFindNextMovie 110, 663
PdrFreeExtension 111
PdrGetCurrentDataset 112, 661
PdrGetCurrentGroup 113, 662
PdrGetExtensionAtPos 114
PdrGetExtensionInfoAtPos 115
PdrGetFirstTcPosition 116, 676
PdrGetFullName 117
PdrGetLastTcPosition 118, 676
PdrGetListOfNames 119, 673
PdrGetMajorVersion 121, 580
PdrGetMaxTracks 122, 700
PdrGetMediaAttributes 123, 699
PdrGetMediaDataset 124, 699
PdrGetMediaDataType 125, 699
PdrGetMediaDuration 126, 699
PdrGetMediaIn 127, 699
PdrGetMediaMarks 128
PdrGetMediaOut 129, 699
PdrGetMediaPath 130, 699
PdrGetMediaStateInfo 131, 707, 708
PdrGetMediaTokenAtPosition 133, 704
PdrGetMediaTokenType 134, 699
PdrGetMinorVersion 135, 580
PdrGetMovieAttributes 136, 659
PdrGetMovieChanges 137, 674
PdrGetMovieCreateTime 139, 659
PdrGetMovieDataset 140, 659
PdrGetMovieGroup 141, 659
PdrGetMovieLastChangeTime 142, 659
PdrGetMovieLength 143, 659
PdrGetMovieMarkIn 144, 659
PdrGetMovieMarkOut 145, 659
PdrGetMovieName 146, 659
PdrGetMovieStateInfo 147, 666, 681
PdrGetMovieTokenFromMediaToken 148, 699
PdrGetNextMediaToken 149, 703
PdrGetNextTrack 150, 701
PdrGetNumMediaOnTrack 151, 702
PdrGetNumMovies 152, 667
PdrGetNumTracks 153, 701
PdrGetPositionOfMediaToken 154, 699
PdrGetPreviousMediaToken 155, 704
PdrGetPreviousTrack 156, 701
PdrGetRegistry 157, 675
PdrGetResourceTypeFromTrackType 158, 706
PdrGetTrackLength 159, 701
PdrGetTrackTokenFromMediaToken 160, 702
PdrGetTrackTokenNum 161, 705
PdrGetTrackTokenType 162, 706
PdrGetTrackTypeFromResourceType 163, 706
PdrGetUserData 164, 671, 672
PdrGetWaitOpStatus 165, 669

PdrInsertExtension 166
PdrMediaTokenIsNull 167, 706
PdrMovieExists 168, 667
PdrNullMediaToken 169, 706
PdrOpenMovie 170, 664
PdrReadExtension 171
PdrRenameMovie 172, 665
PdrRenameOpenMovie 173, 676
PdrRestoreMovieDescription 174, 673
PdrSaveMovie 175, 669, 679
PdrSetClipIn 176, 667
PdrSetClipOut 177, 667
PdrSetCurrentDataset 178, 661
PdrSetCurrentGroup 179, 662
PdrSetMediaIn 180, 704
PdrSetMediaMarks 181, 705
PdrSetMediaOut 182, 704
PdrSetMovieLocked 183, 666
PdrSetMovieMarkIn 184, 679
PdrSetMovieMarkOut 185, 679
PdrSetMovieMarks 186, 679
PdrSetMovieReadOnly 187, 666
PdrSetMovieReadWrite 188, 666
PdrSetMovieUnlocked 189, 666
PdrSetRegistry 190, 675
PdrSetUserData 192, 670, 671
PdrTerminateWaitOperation 193, 670
PdrTrimClip 194, 668
PLS constants 595
PLS serial data types 595
PLS serial definition fields 595
pls_sp.h 595
PlsAddTransport 195, 643
PlsAllocateCartridge 196, 618
PlsAllocateTransport 197, 614
PlsArchiveClip 198, 625
PlsArchiveDataFile 199, 626
PlsBackupCatalog 200, 629
PlsBackupCatalogCommand 629
PlsCancelCommand 201, 645
PlsCleanTransport 202, 617
PlsCloseBinInfo 203, 605
PlsCloseCartridge 204, 618
PlsCloseCartridgeInfo 205, 609
PlsCloseFile 206, 625
PlsCloseFileInfo 207, 611
PlsCloseFindHandle 208, 613
PlsCloseHandle 613
PlsCloseLibrary 209, 598
PlsCloseTransport 210, 614
PlsCloseTransportConfig 211, 603
PlsConnectCartridge 212, 618
PlsConnectFile 213, 625
PlsConnectTransport 214, 614
plsconst.h 595
PlsCopyCartridge 215, 622
PlsDeleteFile 216, 627
PlsExportCartridge 217, 622
PlsFindFirstBinInfo 218, 604
PlsFindFirstCartridgeInfo 219, 606
PlsFindFirstFileInfo 221, 609
PlsFindFirstHandle 223, 611
PlsFindFirstTransportConfig 224, 601
PlsFindNextBinInfo 225, 605
PlsFindNextCartridgeInfo 226, 608
PlsFindNextFileInfo 227, 610
PlsFindNextHandle 228, 613
PlsFindNextTransportConfig 229, 603
PlsFormatCartridge 230, 621
PlsGetAnyEvent 231, 630
PlsGetAsynchEvent 232, 634
PlsGetBackupDir 233, 628
PlsGetBackupDirCommand 628
PlsGetCartDescription 234, 624
PlsGetCartridgeConfig 235, 619
PlsGetCartridgeStatus 236, 620
PlsGetClipSize 237, 629
PlsGetCommandEvent 238, 638
PlsGetEventMask 239, 630
PlsGetFileDescription 240, 627
PlsGetLibraryConfig 241, 599
PlsGetLibraryStatus 242, 600
PlsGetLocationString 243, 623
PlsGetMajorVersion 244, 597
PlsGetMinorVersion 245, 597
PlsGetModes 246, 642
PlsGetPartitionMap 247, 624
PlsGetPath 248, 643
PlsGetStatusCommand 249, 644
PlsGetTimeDate 250, 643

PlsGetTransportStatus 251, 615
PlsHouseKeeping 252, 645
PlsImportCartridge 253, 622
PlsImportLoadCartridge 254, 624
PlsInventoryCartridge 255, 621
PlsInventoryLibrary 256, 644
PlsLoadTransport 257, 617
PlsOpenLibrary 258, 598
PlsRemoveTransport 259, 644
PlsRenameFile 260, 626
PlsRestore 261, 626
PlsRestoreDataFile 262, 628
PlsSetBackupDir 263, 628
PlsSetBackupDirCommand 628
PlsSetCartDescription 264, 623
PlsSetEventMask 265, 629
PlsSetFileDescription 266, 627
PlsSetLocationString 267, 623
PlsSetModes 268, 642
PlsSetPath 269, 642
PlsUnloadTransport 270, 617
PlsUpdateCartridge 271, 621
Profile
    command cycle 564
    command data representation 566
    command fields 564
    command reply structure 565
    command structure 564
    serial command summary 46
Profile commands
    attached media access commands 56, 710
    attached movie access commands 54, 682
    audio commands
        VdrGetAudioInputLevel 318
        VdrGetAudioLevel 319
        VdrGetAudioOutputLevel 320
        VdrGetAudioWindow 321
        VdrSetAudioInputLevel 416
        VdrSetAudioOutputLevel 417
        VdrSetAudioWindow 418
    cut-and-paste commands
        VfsCreateFile 476
    dataset commands
        PdrGetMediaDataset 124
    event access commands 59, 742
    file commands 471
        VfsCreateDirectory 475
        VfsDeleteFile 478
        VfsEraseMediaSegment 480
        VfsFieldWrite 483
        VfsGetCurrentDirectory 488
        VfsMoveFile 505
        VfsSetCurrentDirectory 509
    group commands
        PdrGetMovieGroup 141
    media commands
        PdrFindFirstMediaUsage 104
        PdrFineNextMediaUsage 109
        PdrGetMediaDataset 124
        PdrGetMediaTokenAtPosition 133
        VdrSetMediaUserData 439
        VfsCreateMediaFile 477
        VfsEraseMediaSegment 480
    media token commands
        PdrGetMediaTokenAtPosition 133
        PdrGetMovieTokenOfMediaToken 148
        PdrGetNextMediaToken 149
        PdrGetNumMediaTokensOnTrack 151
        PdrGetPositionOfMediaToken 154
        PdrGetPreviousMediaToken 155
        PdrMediaTokenIsNull 167
    miscellaneous commands
        VdrGetCurrentTimeCode 329
        VdrGetGenTcMode 334, 420
        VdrGetMinPosition 352
        VdrGetPosition 379
        VdrGetVideoFormat 399
        VdrGetVideoGoalSize 397
    movie commands
        PdrDeleteMovie 100
        PdrGetMovieAttributes 136
        PdrGetMovieGroup 141
        PdrGetMovieLastChangeTime 142
        PdrGetMovieLength 143
        PdrGetMovieName 146
        PdrGetMovieStateInfo 147
        PdrGetNextMediaToken 149
        PdrGetNumMovies 152

PdrGetNumTracks 153
movie token commands
PdrCopyMovie 90
PdrCopyMovieToken 91
PdrCreateMovie 96
PdrDeleteMovie 100
PdrGetMovieAttributes 136
PdrGetMovieTokenOfMediaToken 148
PDR movie access commands 52, 658
play commands
VdrGetCurrentRate 327
VdrGetMaxPosition 338
VdrIdle 401
VdrJog 406
VdrSetMaxPosition 434
VdrSetPosition 450
VdrSetPositionLimits 451
PLS access commands 49
port access commands 51, 647
record commands
VdrCueRecord 302
VdrGetMaxPosition 338
VdrSetLumQLimits 431
VdrSetMinPosition 442
VdrShuttle 459
resource access commands 57, 720
resource commands
PdrGetTrackTypeFromResource-Type 163
PdrMediaTokenIsNull 167
VdrAllocateChannel 282
VdrGetChannelInfoList 323
VdrGetMovieAtPosition 354
VdrGetMovieDefaultMarkIn 355
VdrGetMovieDefaultMarkOut 356
VdrGetNumChannelDefs 373
VdrGetPreviousMovie 382
VdrGetResourceFromMedia 388
VdrSetResourceUserData 453
resource token commands
PdrGetNextTrack 150
special recording commands
VdrGetGenTcMode 334
VdrSetChrQLimits 420
state commands
VdrGetCurrentState 328
stored media access commands 55, 699
streaming transfer access commands 62
system access commands 46, 573
track commands
PdrGetNextMediaToken 149
PdrGetNumMediaTokensOnTrack 151
PdrGetNumTracks 153
PdrGetPreviousMediaToken 155
PdrGetPreviousTrack 156
PdrGetTrackLength 159
PdrGetTrackTokenNum 161
PdrGetTrackTokenType 162
PdrGetTrackTypeFromResource-Type 163
PdrMediaTokenIsNull 167
user bit commands
VdrGetCurrentUserBits 330
user data commands
VdrGetMovieUserData 363
VdrGetResourceUserData 391
VdrSetMovieUserData 445
video mix effects access commands 60, 750, 767
ProLink 563

## R

Release Resource 729
RemCloseConnection 272
RemGetConnectionName 273
RemGetMajorVersion 274
RemGetMinorVersion 275
RemGetNodeName 276
RemIsHandleRemote 277
RemIsolateConnection 278
RemIsolateHandle 279
RemOpenConnection 280
Rename Open Movie 676
Reschedule Event 745
Reset Fade 753
Reset Port 657
Reset Resource 729
Reset Transition Type 751

resource lines 570
resource types
ResourceAudioCodec 283
ResourceAudioInput 284
ResourceAudioOutput 284
ResourceDvcpro25Codec 284
ResourceDvcpro50Codec 284
ResourceGenTcInput 284
ResourceGpiInput 284
ResourceGpiOutput 284
ResourceJpegCodec 284
ResourceLtcInput 285
ResourceLtcOutput 285
ResourceMpegDecoder 285
ResourceMpegEncoder 285
ResourceTimecodeRecorder 285
ResourceVideoCombiner 286
ResourceVideoInput 286
ResourceVideoOutput 286
ResourceVitcInput 286
ResourceVitcOutput 286
RS-422 serial communication 563

## S

Schedule Event 744
Set Audio Input Level 734
Set Audio Output Level 735
Set Audio Window 735
Set Bit-Rate 739
Set Border Color 759
Set Border Width 759
Set Chrominance Q Limits 734
Set Current Directory 577
Set Default Tc Format 657
Set Edge Softness 760
Set Encoding Range 740
Set Fade Length 756
Set Fade Matte Color 762
Set Fade Progress 754
Set Field Dominance 657
Set Gen Tc Regen Source 740
Set Horizontal Modulation 760
Set Key 762
Set Luminance Q Limits 733
Set Media Audio Fades 717
Set Media Duration 716
Set Media Mark In 704, 714
Set Media Mark Out 704, 715
Set Media Marks 705
Set Media User Data 716
Set Motion Play/Record Mode 654
Set Movie Mark In 688
Set Movie Mark Out 689
Set Movie Marks 679
Set Movie User Data 690
Set Mpeg Chroma Format 739
Set Mpeg Gop Structure 739
Set Multipliers 757
Set Open Movie Marks 679
Set Position 653, 758
Set Position Limits 655
Set Resource User Data 738
Set Rotation 758
Set Saved Vbi Lines 740
Set Still Play Mode 655
Set Timecode Generator Format 736
Set Timecode Generator Mode 736
Set Transition Length 755
Set Transition Progress 752
Set Transition Type 751
Set Vertical Modulation 761
Set Video Codec Timing 738
Set Video Format 578
Set Video Goal Size 733
Shuttle At Gen Tc 657
Shuttle Play 651
Shuttle Record 652
Start Fade 754
Start Fade At Gen Tc 763
Start Transition 752
Start Transition At Gen Tc 763
State Event 747
State Event At Gen Tc 747
streaming 559

## T

TekCfg library 9, 29, 64
device configuration 9
file system information 9
version information 9

TekPdr library 10, 30, 85
inventory management 10
library management 10
movie characteristics 11
name translation control 11
stored movie editing and evaluation 11
version information 13
TekPls library 14, 33, 195
basic archive operations 14
cartridge management 14
enumeration 15
event management 15
library management 15
miscellaneous commands 15
serial protocol command structure 595
serial protocol commands 595
transport management 16
version information 16
TekRem library 17, 35, 272
connection management 17
version information 17
TekVdr library 18, 36, 281
event management 18
media control 18
movie control 19
port control 20
resource control 21
system information 23
version information 23
TekVfs library 24, 41, 471
copy/move 24
file attributes 24
file enumeration 24
file/directory management 24
read/write 25
version information 25
TekVme library 26, 43, 514
edge and border parameters 26
luminance key parameters 26
matte parameters 26
mix contexts 26
position parameters 26
timing and initial progress parameters 26
transitions and fades 27
version information 27
TekXfr library 28, 45, 559
transfer management 28

## U

UML 562
Uniform Media Locator
(See UML)

## V

VdrActivateResource 281, 731
VdrAllocateChannel 282
VdrAllocateResource 283, 726
VdrAttachFittedMedia 287, 711
VdrAttachFittedMediaWithMarks 288, 712
VdrAttachMedia 289, 711
VdrAttachMediaWithMarks 290, 712
VdrAttachMovie 291, 684, 690
VdrAttachMovieWithMarks 293, 685, 694
VdrAttachOpenMovie 295, 692, 697
VdrAttachOpenMovieWithMarks 296, 695, 698
VdrCancelAllPendingGpiTriggerCommands 297
VdrCancelEvent 298, 737, 745
VdrCancelPendingTimecodeTriggerCommands 299, 657
VdrClosePort 300, 649
VdrCuePlay 301, 650
VdrCueRecord 302, 652
VdrDeactivateResource 303, 732
VdrDefaultEvent 304, 743
VdrDefaultEventAtGenTc 307
VdrDefaultEventOnGpiInputTrigger 308
VdrDetachMedia 309, 713, 737
VdrDetachMovie 310, 686, 737
VdrDisableMoviePositionLimiting 311, 692
VdrEnableMoviePositionLimiting 312, 693
VdrExpandMovie 313, 691, 696
VdrGetAbsMaxChrQ 314, 723
VdrGetAbsMaxLumQ 315, 723
VdrGetAbsMinChrQ 316, 723
VdrGetAbsMinLumQ 317, 723
VdrGetAudioInputLevel 318, 723
VdrGetAudioLevel 319, 721
VdrGetAudioOutputLevel 320, 723

VdrGetAudioWindow 321, 723
VdrGetBitRate 322, 723
VdrGetChannelInfoList 323
VdrGetCurrentFieldSize 324, 721
VdrGetCurrentLumQFactor 325, 721
VdrGetCurrentPictureStatus 326
VdrGetCurrentRate 327, 647
VdrGetCurrentState 328, 647
VdrGetCurrentTimeCode 329, 721
VdrGetCurrentUserBits 330, 721
VdrGetEncodingRange 331
VdrGetFieldDominance 332, 647
VdrGetGenTcFormat 333, 723
VdrGetGenTcMode 334, 723
VdrGetMajorVersion 335, 577
VdrGetMaxChrQ 336, 723
VdrGetMaxLumQ 337, 723
VdrGetMaxPosition 338
VdrGetMediaAtPosition 339, 714
VdrGetMediaAudioFades 340, 717
VdrGetMediaDuration 341, 710
VdrGetMediaEndPosition 342, 710
VdrGetMediaMarkDuration 343, 710
VdrGetMediaMarkIn 344, 710
VdrGetMediaMarkOut 345, 710
VdrGetMediaName 346, 710
VdrGetMediaStartPosition 347, 710
VdrGetMediaUserData 348, 710
VdrGetMinChrQ 349, 723
VdrGetMinLumQ 350, 723
VdrGetMinorVersion 351, 577
VdrGetMinPosition 352
VdrGetMovieAccessMode 353, 682
VdrGetMovieAtPosition 354, 647, 687, 691
VdrGetMovieDefaultMarkIn 355, 682
VdrGetMovieDefaultMarkOut 356, 682
VdrGetMovieEndPosition 357, 682
VdrGetMovieMarkDuration 358, 682
VdrGetMovieMarkIn 359, 682
VdrGetMovieMarkOut 360, 682
VdrGetMovieName 361, 682
VdrGetMovieStartPosition 362, 682
VdrGetMovieUserData 363, 682
VdrGetMpegChromaFormat 364, 723
VdrGetMpegGopLimits 365, 740
VdrGetMpegGopStructure 366, 723
VdrGetNextEvent 367, 746, 747
VdrGetNextMedia 368, 713
VdrGetNextMovie 369, 686
VdrGetNextPort 370, 656
VdrGetNextResource 371, 730
VdrGetNextResourceOfType 372, 730
VdrGetNumChannelDefs 373
VdrGetPlayMode 374, 647
VdrGetPortFromEvent 375, 746
VdrGetPortFromMedia 376, 716
VdrGetPortFromMovie 377, 689
VdrGetPortFromResource 378, 737
VdrGetPosition 379, 647, 656
VdrGetPreviousEvent 380, 746
VdrGetPreviousMedia 381, 713
VdrGetPreviousMovie 382, 687
VdrGetPreviousPort 383, 656
VdrGetPreviousResource 384, 730
VdrGetPreviousResourceOfType 385, 730
VdrGetRecordCapacity 386, 574
VdrGetResourceConnectionHandle 387, 728
VdrGetResourceFromMedia 388, 717
VdrGetResourceNumber 389, 723
VdrGetResourceType 390, 723
VdrGetResourceUserData 391, 738
VdrGetStateChangeLatency 393, 461
VdrGetStillPlayMode 394, 647
VdrGetTotalRecordCapacity 395, 574
VdrGetVideoCodecTiming 396, 723
VdrGetVideoCombinerPartHandle 397, 738
VdrGetVideoFieldCount 398, 580
VdrGetVideoFormat 399, 578
VdrGetVideoGoalSize 400, 723
VdrIdle 401, 650
VdrIsMoviePositionLimitingEnabled 403, 693
VdrIsResourceActive 404, 723
VdrIsResourceAvailable 405, 737
VdrJog 406, 651, 653
VdrMergeResources 407, 739
VdrOpenPortConnection 408, 649
VdrReleaseResource 409, 729
VdrRescheduleEvent 410, 745
VdrResetPort 411, 657

VdrResetResource 412, 729
VdrScheduleEvent 413, 744
VdrSetAudioInputLevel 416, 734
VdrSetAudioOutputLevel 417, 735
VdrSetAudioWindow 418, 735
VdrSetBitRate 419, 739
VdrSetChrQLimits 420, 734
VdrSetDefaultTcFormat 421, 657
VdrSetEncodingRange 422, 740
VdrSetFieldDominance 423, 657
VdrSetFittedMediaMarkIn 424, 714
VdrSetFittedMediaMarkOut 425, 715
VdrSetGenTcFormat 426, 736
VdrSetGenTcMode 427, 736
VdrSetGenTcRegenSource 428, 740
VdrSetGpiOutput 429
VdrSetGpiOutputOnGpiInputTrigger 430
VdrSetLumQLimits 431, 733
VdrSetMaxChrQ 432
VdrSetMaxLumQ 433
VdrSetMaxPosition 434
VdrSetMediaAudioFades 435, 717
VdrSetMediaDuration 436, 716
VdrSetMediaMarkIn 437, 714
VdrSetMediaMarkOut 438, 715
VdrSetMediaUserData 439, 716
VdrSetMinChrQ 440
VdrSetMinLumQ 441
VdrSetMinPosition 442
VdrSetMovieMarkIn 443, 688
VdrSetMovieMarkOut 444, 689
VdrSetMovieUserData 445, 690
VdrSetMpegChromaFormat 446, 739
VdrSetMpegGopStructure 447, 739
VdrSetPlayMode 448, 654
VdrSetPosition 450, 653
VdrSetPositionLimits 451, 655
VdrSetPositionRelative 452, 653
VdrSetResourceUserData 453, 738
VdrSetSavedVbiLines 454, 740
VdrSetStillPlayMode 455, 655
VdrSetVideoCodecTiming 456, 738
VdrSetVideoFormat 457, 578
VdrSetVideoGoalSize 458, 733
VdrShuttle 459, 651, 652
VdrShuttleAtGenTc 460, 657
VdrStateEvent 462, 747
VdrStateEventAtGenTc 463, 747
VdrStateEventOnGpiInputTrigger 464
VdrTriggerGpiOutput 465
VdrTriggerGpiOutputAtGenTc 467
VdrTriggerGpiOutputOnGpiInputTrigger 466
VdrWaitForGpiInputTrigger 468
VdrWaitForVideoField 469, 580
VdrWhenIsEvent 470, 746
Vfs Cancel Copy 590
Vfs Close Handle 587
Vfs Copy File 588
Vfs Copy Media Segment 588
Vfs Create File 583
Vfs Create Media File 583
Vfs Delete File 584
Vfs Erase Media Segment 586
Vfs Field Free 591
Vfs Field Read 590
Vfs Field Read Data 591
Vfs Field Write Commit 592
Vfs Field Write Data 591
Vfs Find Close 588
Vfs Find First File 587
Vfs Find Next File 587
Vfs Get File Status 582
Vfs Get File Time Code 594
Vfs Get Time 593
Vfs Move File 584
Vfs Move Media Segment 589
Vfs Query File System Space 594
Vfs Read File 585
Vfs Set Default Codec Timing 593
Vfs Set File Attributes 585
Vfs Set Time 593
Vfs Status of Copy 589
Vfs Stripe Timecode 592
Vfs Write File 586
VfsCancelCopy 471, 590
VfsCloseHandle 472, 587
VfsCopyFile 473, 588
VfsCopyMediaSegment 474, 588
VfsCreateDirectory 475, 581

VfsCreateFile 476, 583
VfsCreateMediaFile 477, 583
VfsDeleteFile 478, 584
VfsDirectoryExists 479, 581
VfsEraseMediaSegment 480, 586
VfsFieldFree 481, 591
VfsFieldRead 482, 590
VfsFieldWrite 483, 591, 592
VfsFileExists 484
VfsFindClose 485, 588
VfsFindFirstFile 486, 587, 588
VfsFindNextFile 487, 587
VfsGetCurrentDirectory 488, 577
VfsGetCurrentDrive 489, 577
VfsGetEngineMajorVersion 490, 578
VfsGetEngineMinorVersion 491, 578
VfsGetFileAttributes 492, 582
VfsGetFileDefaultMarkIn 493, 582
VfsGetFileDefaultMarkOut 494, 582
VfsGetFileDefaultMarks 495
VfsGetFileModificationTime 496, 582
VfsGetFileRecordedDuration 497, 582
VfsGetFileSize 498, 594
VfsGetFileTimeCode 499, 594
VfsGetFileType 500, 582
VfsGetFileVideoFormat 501, 582
VfsGetMajorVersion 502, 580
VfsGetMinorVersion 503, 581
VfsMoveFile 505, 584
VfsMoveMediaSegment 506, 589
VfsQueryFileSystemSpace 507, 594
VfsReadFile 508, 585
VfsSetCurrentDirectory 509, 577
VfsSetFileAttributes 510, 585
VfsStatusOfCopy 511, 589
VfsStripeTimecode 512, 592
VfsWriteFile 513, 586
video mix effects board 514
VITC 428
VmeAllocateContext 514, 750
VmeApplyContext 515, 750
VmeFreeContext 516, 750
VmeGetAspect 517
VmeGetBorderColor 518, 764
VmeGetBorderColorYUV 519
VmeGetBorderWidth 520, 764
VmeGetEdgeSoftness 521, 764
VmeGetFadeLength 522, 763
VmeGetFadeMatteColor 523, 766
VmeGetFadeProgress 524, 755
VmeGetHModulation 525, 765
VmeGetKey 526, 766
VmeGetMajorVersion 527, 763
VmeGetMinorVersion 528, 763
VmeGetMultipliers 529, 764
VmeGetPosition 530, 764
VmeGetRotation 531, 764
VmeGetTransitionLength 532, 763
VmeGetTransitionProgress 533, 753
VmeGetTransitionType 534, 763
VmeGetVModulation 535, 765
VmeResetFade 536, 753
VmeResetTransition 537, 751
VmeSetAspect 538, 757
VmeSetBorderColor 539, 759
VmeSetBorderColorYUV 540
VmeSetBorderWidth 541, 759
VmeSetEdgeSoftness 542, 760
VmeSetFadeLength 543, 756
VmeSetFadeMatteColor 544, 762
VmeSetFadeProgress 545, 754
VmeSetHModulation 546, 760
VmeSetKey 547, 762
VmeSetMultipliers 548, 757
VmeSetPosition 549, 758
VmeSetRotation 550, 758
VmeSetTransitionLength 551, 755
VmeSetTransitionProgress 552, 752
VmeSetTransitionType 553, 751
VmeSetVModulation 554, 761
VmeStartFade 555, 754
VmeStartFadeAtGenTc 556, 763
VmeStartTransition 557, 752
VmeStartTransitionAtGenTc 558, 763

## W

Wait For Video Field 580
When Is Event 746
Windows NT agreement 3

## X

Xfr Abort 768
Xfr Get Active Tokens 769
Xfr Get Status 768
Xfr Request 767
XfrAbort 559, 768
XfrGetActiveTokens 560, 769
XfrGetStatus 561, 768
XfrRequest 562, 767