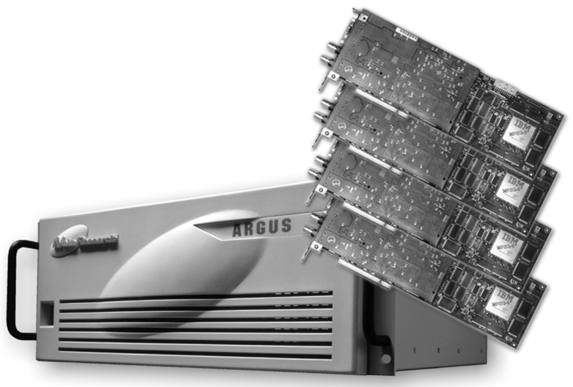
Argus® Multi-Board Encoder

API Developer's Guide

Version 2.61

Application Programming Interface Documentation for the Vela Argus Audio/Video Multi-Board Encoder



MPEG-2 Broadcast Audio/Video Serial Digital Encoding System

Vela®

Document Part Number 9050-1205M

Vela OEM Products Division
5733 Myerlake Circle
Clearwater, FL 33760-2804
Phone: (727) 507-5300
Fax: (727) 573-5310
World Wide Web – http://www.vela.com

Mailing / Shipping Address:
5733 Myerlake Circle
Clearwater. FL 33760-2804

All returns must be accompanied by an authorized RMA number obtained from Vela.

Printed in the United States of America 5-02

Rev. AMA-0205-001

Table of Contents

List of Figures and Tables

Chapter 1

Getting Started

The Multi-Board Encoder

The Vela Argus MPEG-2 multi-board encoder system is a professional audio/video encoding system, hosted by a Microsoft Windows 2000 Platform, that contains two or more Argus 4:2:2 or 4:2:0 single-board encoders. These single-board encoders convert analog and digital audio and video signals into studio-quality MPEG-2 digital streams supporting both Main Profile and (with the 4:2:2 board) 4:2:2 Profile encoding. The resulting MPEG-2 compressed video can then be stored on a hard drive and/or transferred via a network, ultimately to be decoded with an MPEG-2-compliant decoder, similar to the acclaimed Vela CineView Pro, for broadcast or personal viewing.

Vela manufactures the single-board PCI encoders used in the Argus multi-board system. Each board features an IBM MPEG-2 encoder chip to compress and encode video data, and two digital signal processors to compress and encode up to four channels of digital or analog audio.

NOTE: Vela also manufactures the Argus® Spectrum, its multi-stream encoder. The Argus Spectrum is capable of producing up to four encoded streams simultaneously. Typically it is used to generate a production-quality MPEG-2 stream that corresponds to one or more lower-bitrate streams. For instructions on programming the multi-stream encoder, please refer to the *Argus Spectrum API Developer's Guide*.

The Multi-Board API

Designed using an object-oriented approach, the Application Programming Interface (API) for the Argus multi-board system is similar to Versions 2.3 and later of the single-board Argus API. As a developer, you need only to create a multi-board Filter Manager object for each active encoder board, then issue calls to the Filter Manager interface to initialize, cue, start, stop, pause, and resume the encode.

Each core function of the encoder has its own COM (Microsoft's Component Object Model) component, also referred to as a filter. One instance of the Filter Manager is responsible for managing all of the filters, in effect controlling the encoding sessions for a specific encoder board, beginning with the reading of

NOTE: "Argus" and "CineView" are registered trademarks of Vela LP. All other trademarks, brand names, or product names appearing in this publication are registered to their respective owners.

the encoding parameters from the Windows Registry through to the storage or transmission of the last byte of encoded material.

What's New Since Version 2.5

New Features

- Support for Dolby* Digital (AC-3) encoding has been added as an optional feature. Ask your Vela sales representative for more information.
- Audio channels 3 and 4 start more reliably.
- The length of time required to cue has been reduced.

Filter Manager automatically detects and sets drop-frame mode if VTR Control is enabled.

Figure 1-1 illustrates the configuration of Argus filters used to control an encoding session performed by one specific encoder board.

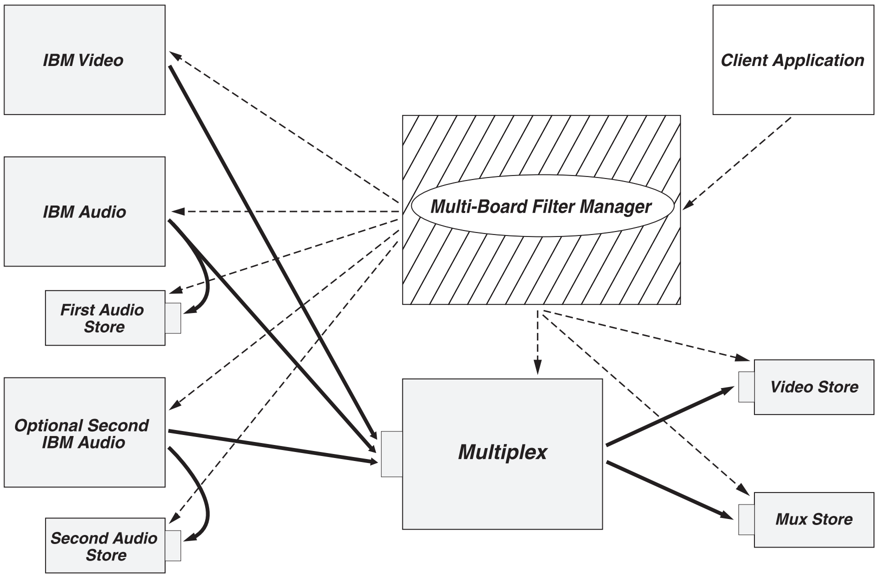


Figure 1-1. Argus Filter Architecture for Control of a Single Encoder Board

*Dolby is a trademark of Dolby Laboratories.

In Figure 1-1:

- The solid lines represent data flow.
- The broken lines represent command flow.
- The gray rectangles are filters (COM components that send or receive MPEG data).
- The IBM Video filter reads video data from the encoder board and delivers it to the Multiplex component
- The IBM Audio filter reads audio data from the encoder board and delivers it to the Multiplex component.
- The Multiplex filter receives audio and video data from the IBM Audio and IBM Video filters, optionally processing the video stream to insert closed captioning or to adjust GOP header time codes. It then performs one of the following actions:

    Merges the audio and video data together into a single system, program, or transport stream, delivering the single stream to a Remote Store filter, referred to as MuxStore.

    Delivers the video data to a Remote Store filter, referred to as VideoStore. When you elect this option, the IBM Audio filters deliver their audio data directly to FileStore filters known as FirstAudioStore and (optionally) SecondAudioStore. The audio filters are able to store their streams directly to files because, unlike the video elementary stream, the audio streams require no additional processing.

- MuxStore (or VideoStore, FirstAudioStore, and SecondAudioStore) writes its data to a file.
- FilterManager controls all of the filters. In response to commands that it receives from your client application, the Filter Manager component issues commands to all of the filters and receives responses from them in the form of COM events. It communicates with the client application by sending log events, error events, and finished events.

What's Different?

As we mentioned previously, the multi-board encoder API is similar to the standard single-board Argus encoder API, with a few major differences:

- You can simultaneously run multiple instances of the multi-board Filter Manager interface, each responsible for the operation of a single Argus encoder board. Before calling the Initialize() method for each instance of the Filter Manager, just call the PutEncoderBoardNumber() method to indicate which board that instance of the Filter Manager is to control.

- Each of the Windows Registry tables that store encoding parameters is uniquely numbered to indicate to which encoder board the parameters apply. For example, for encoder board 0, the multiplex (or mux) parameters are now stored in the Windows Registry table named Mux0; those for board 1 are stored in Mux1; and so on.
- Where there are multiple output files (for example, when producing audio and video elementary files), you can set the properties for each output file separately, using separate Registry tables.
- The multi-board API does not currently support real-time playback (confidence monitoring), multi-stream-encoding, or any of the other features that require the use of a CineView Pro family decoder.
- The multi-board API does not currently support VTR control. If you wish to control a VTR as part of the encoding process, you might want to use the VTR API, detailed in Chapter 3, to monitor the position of the tape in preparation for starting the encode. As a result of this change, the duration of the encode is now set in the Mux table.

Minimum System Requirements

- Windows 2000 operating system, with Service Pack 2.
- Microsoft Internet Explorer 5.0
- IBM PC or PC-compatible Pentium 600 MHz dual-processor system with PCI bus.
- 256 MB RAM minimum
- CD-ROM drive (for installation of system files)
- One or more Vela Argus 4:2:2 or 4:2:0 encoder boards. (Can be mixed)
- 9.0 GB hard drive (Fast/Wide SCSI recommended)

Software Requirements

- Argus multi-board encoder software installation (version 2.6)
- Argus multi-board encoder SDK installation (version 2.6)
- Compiler with COM support (Microsoft Visual C++, with service pack 4 or higher, or Visual Basic 6 is recommended.)

Included Files

The following table is a list of all files to be installed with the standard installation and with the installation of the SDK. Those that are installed as part of the SDK are located in the folder C:\Program Files\Vela Research\Argus\SDK or in one of its sub-folders.

Argus Multi-Board SDK Included Files		
Filename	**File Description**	**Folder**
MultiBoardFilterMgrU.dll IBMAudioMT.dll IBMVideoMT.dll FileStoreMT.dll MultiplexMT.dll ArgusPlugInMT.dll (optional)	Argus multi-board COM components.	C:\Program Files\Vela Research\Argus\SDK_MB
MultiBoardServer.exe	Optional COM component that may be used **in place of** MultiBoardFilterMgrU.dll to run as an out-of-process server. To use, just change (#import) statement with “MultiboardServer.tlb” instead of “MultiBoardFilterMgr.tlb.” (See StdAfx.h.) Also, you must register MultiBoardServer.exe instead of MultiBoardFilterMgrU.dll	C:\Program Files\Vela Research\Argus\SDK_MB
MemMgrMT.dll MemMgrServerMT.exe PinsMT.dll	Modules used for communication among COM components.	C:\Program Files\Vela Research\Common
MBProps.exe MBProps.ico	Application for editing encoder parameters in the Windows Registry.	C:\Program Files\Vela Research\Argus\SDK_MB
MultiBoardFilterMgr.tlb MultiBoardServer.tlb	Type libraries for Filter Manager COM components. Select one of these to correspond to the component you will be using.	C:\Program Files\Vela Research\Argus\SDK_MB\TypeLibs
MBTestApp.dsp Associated source code	Work space containing source code for the C++ sample application that drives a single encoder board.	C:\Program Files\ Vela Research\ Argus\ SDK_MB\ FourBoardTestApp

Table 1-1. Argus Multi-Board SDK Included Files

Argus Multi-Board SDK Included Files (Continued)		
Filename	**File Description**	**Folder**
MBProps.dsp Associated source code	Work space containing source code for the C++ sample application that manages the encoder parameters in the Windows Registry.	C:\Program Files\Vela Research\Argus\SDK_MB\MBProps
FourBoardTestApp.dsp Associated source code	Work space containing source code for the C++ sample application that drives up to four encoder boards simultaneously.	C:\Program Files\Vela Research\Argus\SDK_MB\FourBoardTestApp
VTRTestApp.dsp Associated source code	Work space containing source code for the Visual C++ sample application that controls a tape deck.	C:\Program Files\Vela Research\Argus\SDK\VTRTestApp
sbencode.exe sbetest.exe sbe.exe	Applications that can be used with Customer Support to diagnose problems with the encoder hardware.	C:\Program Files\Vela Research\Argus\Diags
Various	System DLLs for ATL and MFC. See "Distributing Components."	C:\Winnt\System32

Table 1-1. Argus Multi-Board SDK Included Files (Continued)

Component Summary

The goal of this API set is to give you, the Visual Basic, Visual C++, or Java developer, an easy-to-use interface to the Argus encoder. As described earlier in this chapter, we have implemented the encoder software on the Windows 2000 platform as a set of COM (Component Object Model) components.

The only component with which your software must interface directly is the multi-board Filter Manager (**MultiBoardFilterMgrU.dll** or **MultiBoard-Server.exe**). You'll use the interface to the Filter Manager to initialize, cue, start, stop, pause, resume, or reset the encoder board.

Note that, beginning with this release (2.6), you can choose one of two Filter Manager components: the in-process MultiBoardFilterMgrU.dll component, or the out-of-process MultiBoardServer.exe component. Both have identical interfaces, so you need to do just two things to identify the particular component you wish to use: (1) name its .tlb file in the (#import) statement within the StdAfx.h; and (2) register the component itself.

In preparing to encode a clip, your application must configure a set of encoding parameters using the Windows Registry. With the SDK, we provide the code for a sample application, MBProps, which demonstrates the setting of parameters using the Registry. Using the CRegistry class provided with MBProps, you should be able to write code to customize your encoding environment.

The multi-board Argus SDK also includes an auxiliary API that can be used to control a tape deck independently of an encoding session. This second component, VTR.dll, is can be found under “COM Components,” page 58.

In developing a client application to control the encoder, you'll need to follow these steps:

1. Determine which encoding properties will most likely remain unchanged from one encoding session to another. These properties can be set using MBProps (or a similar application). For a complete list of multi-board encoding properties, see Appendix A. For a description of the MBProps application, refer to “MBProps,” page 39.

2. Create a project that supports COM. Within the project, create an instance of IMBFilterMgr (the multi-board Filter Manager interface) and of CFilterManagerEvents (its events interface) for each encoder board that your application will control. After setting the board number, call Initialize() for each of the instances of the Filter Manager. (See “Sample Applications,” page 28).

3. For each instance of the Filter Manager, before cueing for an encode, use the CRegistry class to set any encoding parameters that need to be changed for the upcoming encode (for example, file name or duration).

4. For each instance of the Filter Manager, call Cue() to set up for the encoding session. Then call Start() to start the encoding session.

5. In the method that receives the finished event from the Filter Manager, add code that resets your application when the clip has finished encoding.

For a more detailed description of the steps required to create a client application, refer to “Sample Applications,” page 28. For a complete listing of all of the multi-board encoding parameters, refer to Appendix A.

The COM components that drive the Argus multi-board encoder are all self-registering. The multi-board Argus installation program registers each component using **regsvr32.exe**, a utility that is included with the Argus API distribution. This utility can be used to remove components from the Registry, to add new components, or to replace existing components with newer versions.

SDK Installation

Note that the installation of the Argus Software Developer's Kit is a **password-protected** process. Included with the SDK is an authenticated password that allows installation of the SDK and accompanying files. If you did not receive a password with your SDK purchase, contact Vela Support.

If a previous version of Argus encoder system software is installed on your system, it **must** be uninstalled before continuing with the installation of the version 2.6 SDK. Use the Windows Control Panel "Add/Remove Programs" function to uninstall Argus software, if necessary.

If you have **not** already installed version 2.6 of the Argus system software, you must refer to the Argus/CineView Pro installation instructions for Windows NT or Windows 2000 to install the software before continuing. See the appropriate product installation and user manual for complete instructions.

If you have already installed version 2.6 of the Argus software on your system, but did not check the "Argus SDK" during the first installation, you can install it at this time.

To begin the installation, simply insert the Argus system software CD-ROM and, from the Autorun setup screen (Figure 1-2), select "Install Argus or Argus Spectrum", and follow the steps below:

1. Read the "Welcome" screen (Figure 1-3), then click Next.
2. On the "Choose Destination Location" screen (Figure 1-4), accept the C:\Program Files\ Vela Research destination, as listed, by clicking Next. **Do not change the destination, as it is important for proper system operation.**
3. On the "Select Components" screen (Figure 1-5):
   - Under SDK, check the "Argus and Argus Spectrum SDK" check box. (Because it is password-protected, you will be able to install the SDK only if you purchased it and received the corresponding password. If you cannot locate the password, call Vela Support for assistance.)
   - If you have not already done so, you **must** run the "MFC Update" and "Core Encoder Modules" (under Required Components on the "Select Component" screen). These two check boxes **must** always be checked to insure proper installation of the SDK software.
   - Click Next to proceed with the installation of the selected components.
4. From the "Select Program Manager Group" screen (Figure 1-6), accept "Vela Research" by clicking Next.

5. On the “Start Installation” screen (Figure 1-7), click Next.
    - A “DO NOT REMOVE THE CD” screen will display as a reminder that a number of reboots may be required during the installation process, click OK.
    - If you have chosen to run the MFC Update option, the installation process will begin here to copy files.
    - An “Install” message box will appear advising that the system must be restarted. Click OK, and then wait as the system reboots. **Leave the CD-ROM in the drive through the restart process.**

6. If you remembered to leave the CD-ROM in the drive, the setup application pops up immediately after the system reboot. Continue with the installation by following these steps:
    - Select the “I Agree” radio button on the “Argus SDK End User License agreement” screen (Figure 1-8). Click OK.
    - On the “Password” screen (Figure 1-9), you will be asked for a password. Use the one supplied with your SDK. If you have problems finding your password, contact Vela Support (see page 15). After entering the password, click OK.
    - At this time, the application will install some files.

7. On the “Installation Complete” screen (Figure 1-10), note that Argus 2.6 has been successfully installed. Click Finish.

8. The “Install” message box will appear advising that the system must be restarted. Click OK then let the system reboot. **Leave the CD-ROM in the drive through the system restart process.**

9. After the system has rebooted, close the setup application if it is active, then remove the CD-ROM from the drive.

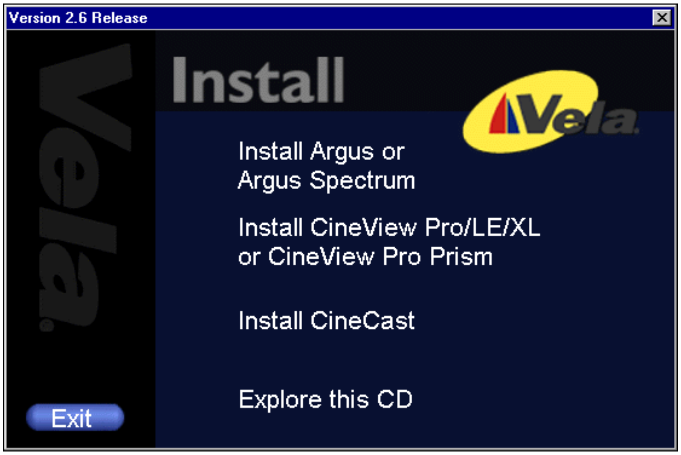
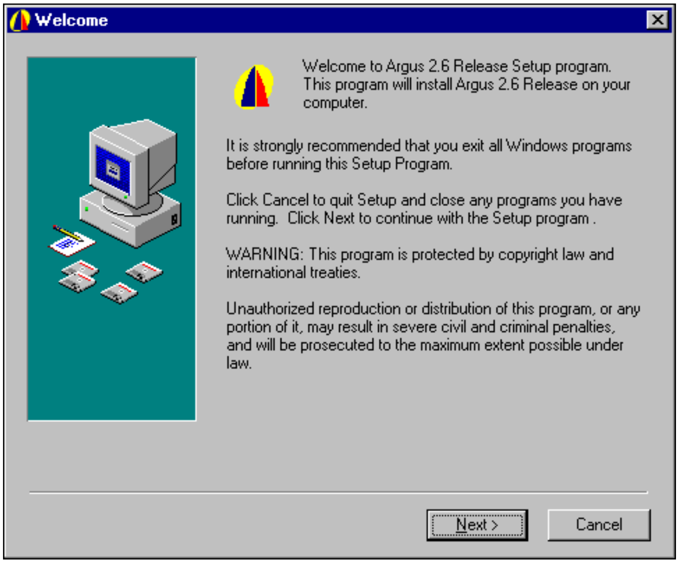

Version 2.6 Release
Install
Vela
Install Argus or Argus Spectrum
Install CineView Pro/LE/XL or CineView Pro Prism
Install CineCast
Explore this CD
Exit

Figure 1-2. Installation Autorun Screen



Welcome to Argus 2.6 Release Setup program. This program will install Argus 2.6 Release on your computer.

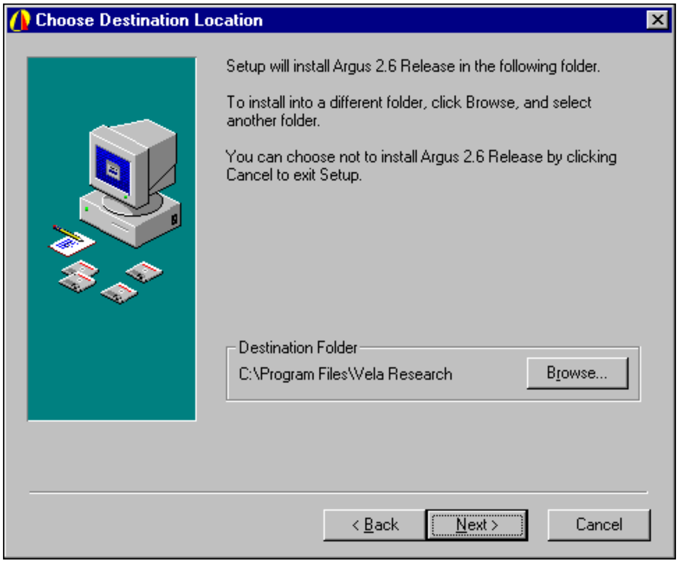
It is strongly recommended that you exit all Windows programs before running this Setup Program.

Click Cancel to quit Setup and close any programs you have running. Click Next to continue with the Setup program .

WARNING: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.

Figure 1-3. Installation Welcome Screen



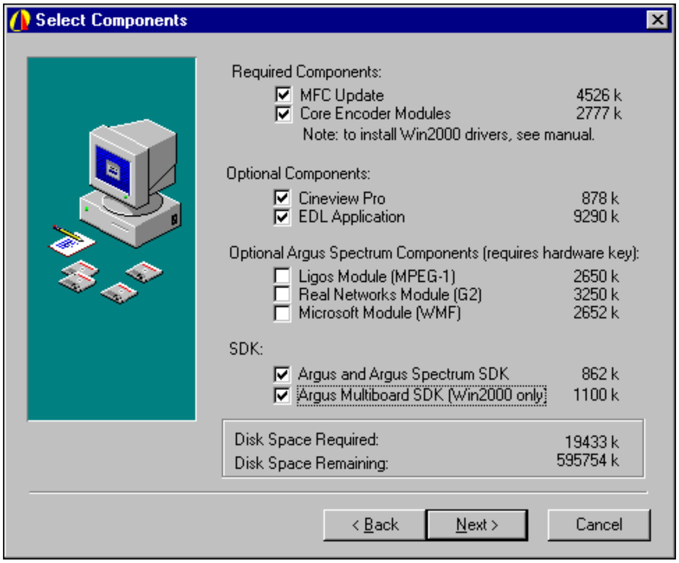
Setup will install Argus 2.6 Release in the following folder.

To install into a different folder, click Browse, and select another folder.

You can choose not to install Argus 2.6 Release by clicking Cancel to exit Setup.

Destination Folder
C:\Program Files\Vela Research

Figure 1-4. Destination Location Screen



Required Components:

[✓] MFC Update	4526 k
[✓] Core Encoder Modules	2777 k

Note: to install Win2000 drivers, see manual.

Optional Components:

[✓] Cineview Pro	878 k
[✓] EDL Application	9290 k

Optional Argus Spectrum Components (requires hardware key):

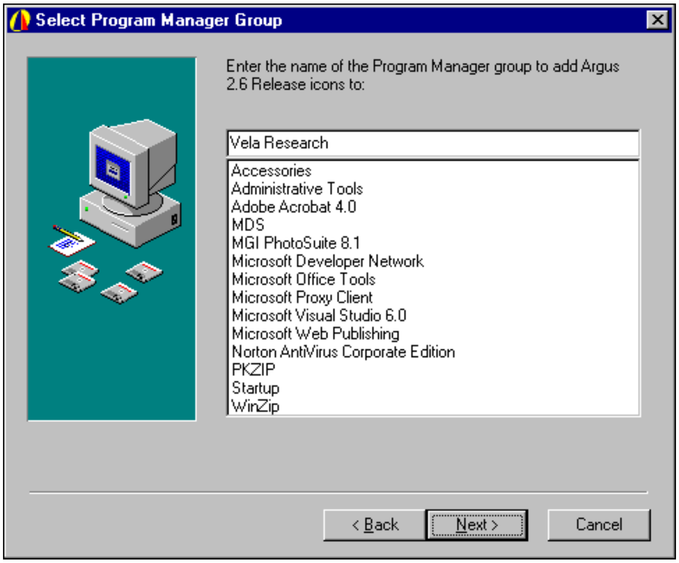
[ ] Ligos Module (MPEG-1)	2650 k
[ ] Real Networks Module (G2)	3250 k
[ ] Microsoft Module (WMF)	2652 k

SDK:

[✓] Argus and Argus Spectrum SDK	862 k
[✓] Argus Multiboard SDK (Win2000 only)	1100 k

Disk Space Required:	19433 k
Disk Space Remaining:	595754 k

Figure 1-5. Select Components Screen

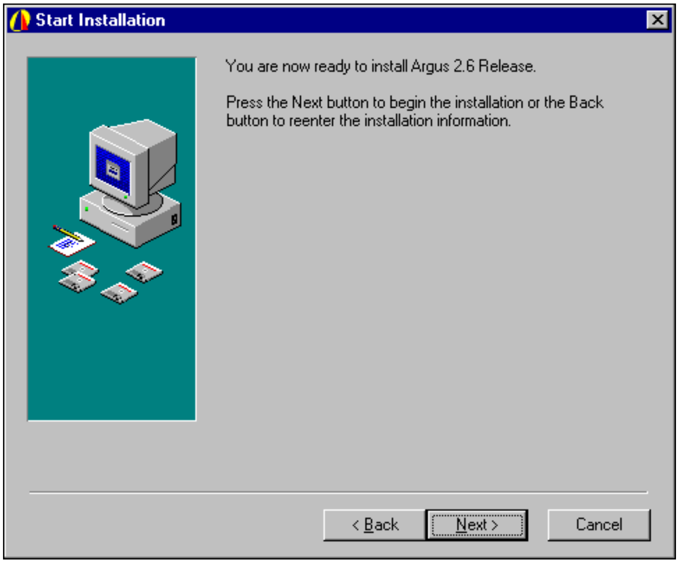

Select Program Manager Group

Enter the name of the Program Manager group to add Argus 2.6 Release icons to:

Vela Research

Accessories
Administrative Tools
Adobe Acrobat 4.0
MDS
MGI PhotoSuite 8.1
Microsoft Developer Network
Microsoft Office Tools
Microsoft Proxy Client
Microsoft Visual Studio 6.0
Microsoft Web Publishing
Norton AntiVirus Corporate Edition
PKZIP
Startup
WinZip

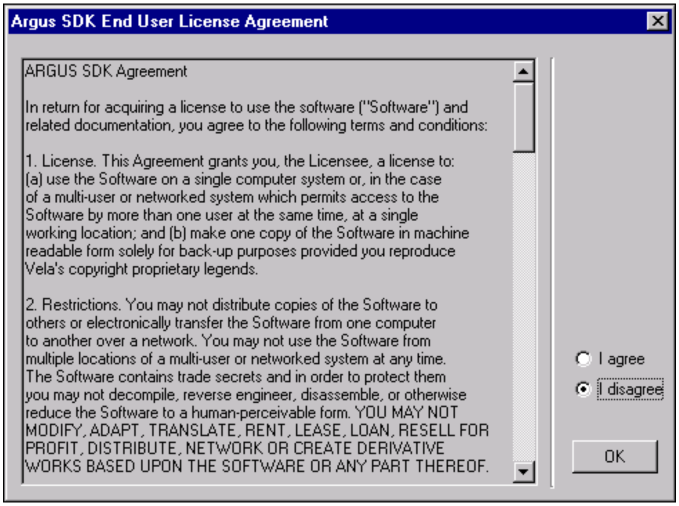
Figure 1-6. Select Program Manager Group Screen


Start Installation

You are now ready to install Argus 2.6 Release.

Press the Next button to begin the installation or the Back button to reenter the installation information.

Figure 1-7. Installation Start Screen


Argus SDK End User License Agreement

ARGUS SDK Agreement

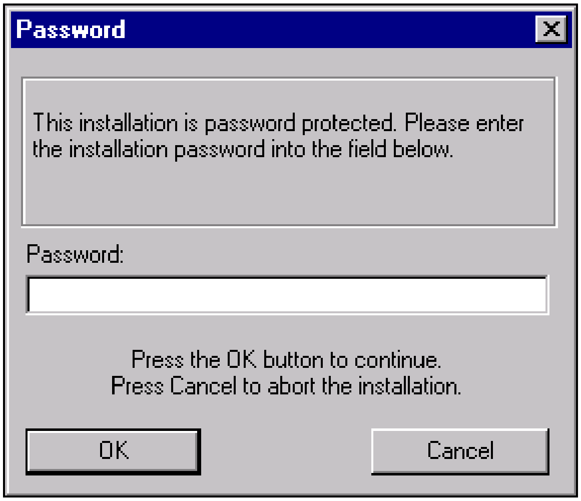
In return for acquiring a license to use the software ("Software") and related documentation, you agree to the following terms and conditions:

1. License. This Agreement grants you, the Licensee, a license to: (a) use the Software on a single computer system or, in the case of a multi-user or networked system which permits access to the Software by more than one user at the same time, at a single working location; and (b) make one copy of the Software in machine readable form solely for back-up purposes provided you reproduce Vela's copyright proprietary legends.

2. Restrictions. You may not distribute copies of the Software to others or electronically transfer the Software from one computer to another over a network. You may not use the Software from multiple locations of a multi-user or networked system at any time. The Software contains trade secrets and in order to protect them you may not decompile, reverse engineer, disassemble, or otherwise reduce the Software to a human-perceivable form. YOU MAY NOT MODIFY, ADAPT, TRANSLATE, RENT, LEASE, LOAN, RESELL FOR PROFIT, DISTRIBUTE, NETWORK OR CREATE DERIVATIVE WORKS BASED UPON THE SOFTWARE OR ANY PART THEREOF.

I agree [✓ I disagree]

Figure 1-8. License Agreement Screen

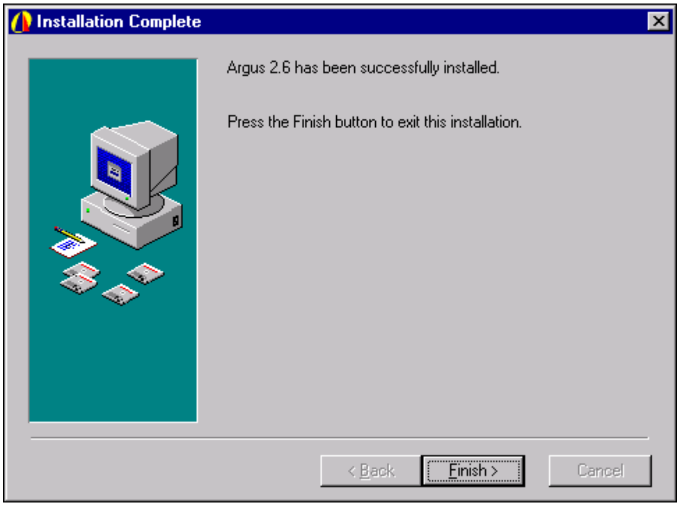

Password

This installation is password protected. Please enter the installation password into the field below.

Password:

Press the OK button to continue.
Press Cancel to abort the installation.

Figure 1-9. Password Entry Screen



Argus 2.6 has been successfully installed.

Press the Finish button to exit this installation.

Figure 1-10. Installation Complete Screen

Suggested Reading

This manual assumes that you are familiar with ATL, COM, and C++ (or Visual Basic) programming. For more information on these topics, we recommend that you refer to the following publications.

ATL/COM References

Inside COM, Rogerson; Microsoft Press. Recommended for COM introduction. Covers COM programming explicitly from a C/C++ hard-core, low-level mode.

ATL Internals, Rector and Sells; Addison-Wesley. This is an excellent reference for ATL programming using Visual Studio 6.0. It includes very useful sections on Smart Pointers, BSTRs, and events.

ATL COM, Grimes, Stockton, Reilly, and Templeton; WROX Press. This book delves deep into the heart of the Active Template Library. Primarily deals with server-side issues, but has some client code development considerations as well.

C++ References

The C++ Programming Language, Stroustrup. This bottom line reference on the C++ programming language is highly recommended for the serious developer.

Using Visual C++, Gregory; QUE Publishing. A comprehensive reference for Microsoft's VC++ compiler.

Other References

References on the Sony® 9-Pin Protocol, used for VTR machine control, are available on the Internet or by contacting Sony Broadcast and Professional Company (Division of Sony Corporation).

Customer Support

In the event of questions or problems with Vela Application Programming Interface methods, materials, or this manual, do not hesitate to contact Vela Training and Support as follows:

- Phone: (727) 507-5301
- E-mail: support@vela.com

World Wide Web - http://www.vela.com

Chapter 2

Using the Multi-Board Encoder API

Overview

The key element of Version 2.6 of the multi-board Argus API is the Filter Manager COM component, which offers two custom interfaces. The primary interface allows your application to make requests of the Filter Manager. The second custom interface (the outgoing interface) allows the Filter Manager to send COM events to your calling application. Note that both Filter Manager interfaces use Unicode-style character strings.

Multi-Board Filter Manager Interfaces

The Primary Interface

The primary Filter Manager interface exposes methods that service requests for encoder functionality. Specifically, it accepts requests to initialize and reset the encoder software, as well as requests to cue, start, stop, pause, and resume an encoding session. Additionally, it exposes methods to read hardware and firmware version numbers, to calculate useful time codes, and to track the status and progress of an encode.

Through its primary interface, the Filter Manager exposes methods and properties. If you are unfamiliar with methods and properties, or with other aspects of object-oriented programming, take time to review reading material on C++ or COM. Refer to the end of Chapter 1 for some suggestions.

A *method* is simply a function call. Usually this function performs an operation, then returns a status code. Each of the fully supported Filter Manager methods is defined in this manual. The definition includes a description of the operation that the method performs as well as a list of the possible return values. Be sure to check the return value of any method that you call before proceeding with the encoding process.

Similar to a C++ class data member, a *property* has a value or a setting. Typically the value of the property can be set by calling a specific Put() method exposed by the Filter Manager interface. Likewise, its value can be retrieved with a Get() method. However, in this version of the multi-board Argus software, we concentrate more on the Windows Registry and less on COM properties to set and retrieve encoding parameters. In fact, the multi-board Filter Manager interface exposes just four fully supported properties.

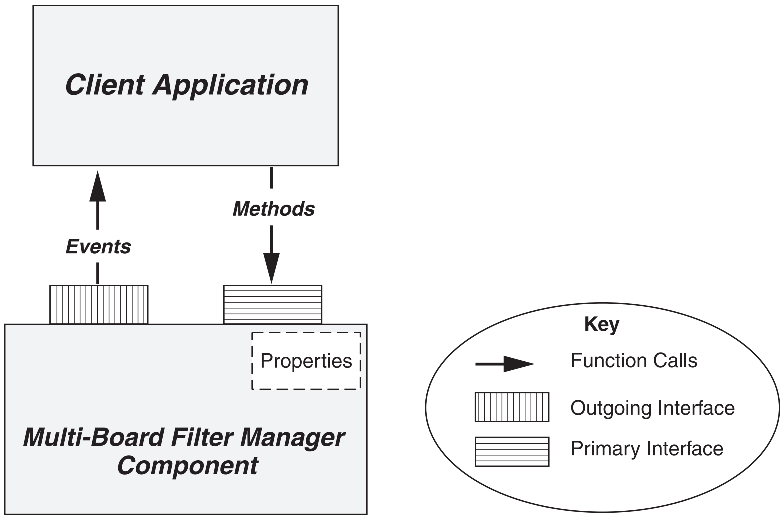


Figure 2-1. Filter Manager Interfaces

The Secondary (Outgoing) Interface

Through its outgoing interface, the multi-board Filter Manager component implements events. An event is a COM mechanism that allows the component to send messages to the calling application. The Filter Manager uses events to issue log messages, error messages, pause / resume messages, and finished messages to the client application. The client can register to receive these messages, at your discretion.

The remainder of this section describes techniques for setting encoding parameters, then defines and describes each of the encoding commands exposed through the primary Filter Manager interface.

Common Encode Parameters: The Windows Registry

Configurable parameters for the multi-board encoder are stored in the Windows Registry. Many of these properties will probably be set once, when the encoder software is installed, and never changed. It is advisable to use an application similar to MBProps (whose source code is included with the SDK) to set these "fixed" properties. Other properties (for example, the file name and duration) will probably change with each encoding session. Using the CRegistry class provided with the SDK, your application can easily set the Registry keys for these properties.

Each of the sub-components controlled by the Filter Manager has its own Registry table, from which it loads and stores encoding properties. The final character of the table name is a single digit, specifying the board to which the properties apply. For example, the multiplex properties for board #0 are stored in the Registry table Mux0.

Figure 2-2 illustrates typical interactions between Argus-related software and the Windows Registry.

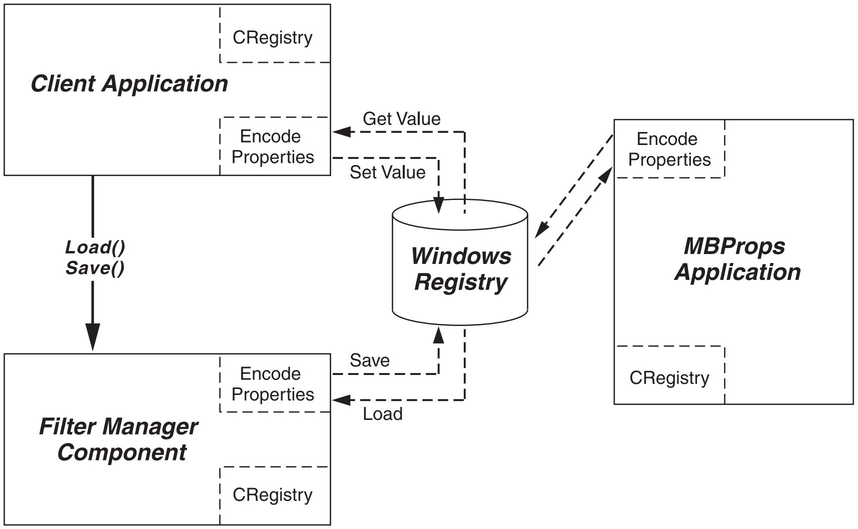


Figure 2-2. Windows Registry Transactions

One useful feature of the Registry method of storing encoding parameters is that it is not necessary to set up the Registry before the first encode. If the application attempts to load from the Registry when there are no encode settings there, the Filter Manager responds by saving all of the default settings to the Registry, creating all of the needed keys. You can then programmatically or manually (using Microsoft's regedit or regedt32 tool) change Registry settings before subsequent encodes.

All of the Registry settings for the multi-board Argus are stored in one of five Registry locations under HKEY_CURRENT_USER \ Software \ Vela Research \Broadcast Argus. These sub-locations are: MultiBoardFilterMgrX, IBMAudioX, IBMVideoX, MuxX, and RemoteStoreX. In each case, the 'X' represents the board number. Appendix A identifies and describes in detail each of the Registry settings that support the Argus multi-board encoding process.

There are a number of circumstances in which you may need to access the encoding parameters that are stored in the Windows Registry. As shown in Table 6-1, the Argus SDK provides a variety of tools to assist with the task of managing the encode parameters.

Task	Tool	Description
Review, modify, save the full set of parameters through an application other than the user interface.	MBProps	An application that displays all of the encoding parameters, allowing the user to review and modify them. Source code is provided with the SDK. See “MBProps,” page 39.
Change individual Registry settings (such as file name or duration) before an encode.	CRegistry class, SetValue() method.	The CRegistry class, whose source code is provided with the MBProps application, provides easy-to-use GetValue() and SetValue() commands to manage encoding parameters of all data types.
Display a specific set of encode parameters through the user interface to the encoder	CRegistry class, GetValue() method.	
Load the full set of encoding parameters from the Windows Registry in preparation for an encode.	MultiBoardFilterMgr Load() method.	Loads all of the encode parameters from the Registry into the specific encoder COM components to which the parameters apply.
Save the full set of encoding parameters under which Argus is currently encoding.	MultiBoardFilterMgr Save() method.	Dumps all of the encoding parameters currently in memory out to the Windows Registry.

Table 2-1. Managing Encode Parameters

Changing Individual Registry Settings

The multi-board encoder SDK contains source code for a sample application, MBProps, that illustrates the loading, modification, and saving of each of the individual Registry settings. This source code is installed in the C:\Program Files\Vela Research\Argus\SDK\MBProps folder. You'll probably find that you can configure most of the encoding properties once using a Registry editing tool like the MBProps application, then leave the settings untouched.

However, a few properties (for example, the output file name and the duration of the encode) are likely to change with each encode. Using the source code of the MBProps application as an example, you can programmatically change these settings before loading and cueing for each encoding session.

MBProps includes source code for a Registry-management class, CRegistry, that makes it easy for you to access and change Registry settings.

MultiBoardFilterMgr Registry-Access Methods

The Filter Manager interface exposes a Load() method that loads the full set of encode parameters from the Registry into memory, as well as a Save() method that writes all the encoder's current property settings to the Registry. Before calling the MultiBoardFilterMgr Cue() method to set up for an encode, you should first write to the Registry any individual property changes that you need to make, then call Load() to load all of the encoder settings into memory. Refer to the source code of our sample application, MBTestApp, for an example. You can preserve the current encoder settings by calling Save() with each encode. It is wise to include this step following a successful call to Cue(), as the cue method may modify several of the Registry settings to meet encoding requirements.

For example, it may reset the video GOP size to agree with the GOP structure and the GOP open/close setting. Calling Save() following a successful cue guarantees that corrections to the encoding parameters are stored in the Registry.

long Load() – Loads all of the settings from the Registry. If the Registry table does not yet exist, the Load() call creates it, enters all of the Registry keys, and assigns their default values. The Load() method returns 0 if it is successful, or a negative error code if it fails. Definitions of each of the error codes are listed in Appendix B.

long Save() – Saves to the appropriate Registry keys all of the encoding parameters that are currently in memory. It returns 0 if the save procedure finished successfully or a negative error code if it failed.

MultiBoard FilterMgr Interface Properties

Because most of the encoding parameters are stored in the Windows Registry, this section is short. In fact, apart from the encoder commands described in the next section, MultiBoardFilterMgr exposes only two methods and four properties, as listed on the following page:

Methods:

long GetDurationFrameCount() – Anytime after the Cue() method has been called for the current encode, GetDurationFrameCount() can be called to retrieve the total number of frames that are to be encoded. This number is calculated during the Cue() process. It can be used in combination with the current frame count to determine what percentage of the encode has finished.

long GetCurrentFrameCount() – While an encode is in progress, this property indicates the number of frames that have already been encoded. Specifically, it identifies the number of picture headers that the MuxMT filter has received from IBMVideoMT and parsed. It has no meaning if an encode is not currently running. If this property is used to update a progress bar, it should be called in its own separate thread.

Properties:

short EncoderBoardNumber – The PutEncoderBoardNumber() method assigns <val> as the encoder board number for this Filter Manager object. The encoder board number must be assigned before your application calls Initialize(). The default value of the board number is 0 (where 0 specifies the first encoder board). The GetEncoderBoardNumber() method is also supported, returning the board number represented as a short.

BSTR EncoderHardwareVersion – The GetEncoderHardwareVersion() method returns the string representation of the hardware version of the encoder board that this MultiBoardFilterMgr component controls. "S422 SBE" is the 4:2:2 single-chip encoder board, and "S420 SBE" is the 4:2:0 single-chip encoder board.

BSTR EncoderFirmwareVersion – The GetEncoderFirmwareVersion() method returns the version-number character string of the firmware installed on the encoder board that this MultiBoardFilterMgr component controls. Only version "1.20" and later support closed captioning. Firmware versions "3.0" and later support configurable reference levels (+4 dB, 0 dB, and -10 dB) for each of the two audio streams.

BSTR SecondAudioVersion – GetSecondAudioVersion() returns the version number character string of the firmware that controls the second audio board managed by this instance of MultiBoardFilterMgr.

MultiBoardFilterMgr Commands

In previous sections we discussed methods and classes used to load and save encoding properties. The multi-board Filter Manager exposes another set of methods, referred to as commands, that you can use to start, stop, and otherwise control an encoding session. These same commands are exposed through the API of all Argus encoders. Each of these methods performs a specific action, then returns a result. When the requested action is performed successfully, the returned result is always 0. When the method fails, it returns a negative result. To determine the nature of the failure, reference Appendix B of this manual, where all of the Argus error codes are listed and explained.

Before performing the action requested by a command, the multi-board Filter Manager checks a state table, summarized below, to determine if the requested action is legal:

Allowable State Transitions			
Current State	**Allowed Commands**	**Resulting State**	
		Success	**Failure**
Start State	Initialize()	Initialized	Exit
	Exit		
Initialized	Reset()	Reset state	Exit
	Exit		
Reset State	Cue()	Cued	Initialized
Cued	Start()	Started	Initialized
Started	Pause()	Paused	Initialized
	End() or Stop()	Initialized	Initialized
Paused	Resume()	Resumed	Initialized
Resumed	Pause()	Paused	Initialized
	End() or Stop()	Initialized	Initialized

Table 2-2. Argus Allowable State Transitions

Multi-Board Encoder Commands

Note that all of the commands listed below are board specific. Once a Filter

Manager object is created, you should immediately call the PutEncoderBoardNumber() method to bind the Filter Manager to a specific encoder board. Then call Initialize(). If Initialize() returns successfully, use the state table above to determine the allowable sequences of encoder commands.

long Initialize() – Sets up the encoder application, binding the current instance of the Filter Manager to a specific board, creating an instance of each board-specific COM object, initializing the drivers, and resetting the encoder board. Must be called after a call to PutEncoderBoardNumber(), defined in the previous section of this manual. A return of 0 indicates that the Filter Manager was set up successfully and that it is now safe to call Cue().

long Cue() – Should be called after Load() returns with a value of 0, indicating valid encoder settings were loaded from the Windows Registry. Cue() sets up each component for an encode, based on the encoding parameters that apply. Generally speaking, it communicates all of the requested encoder settings to the audio and video encoders and opens requested output files. If the Filter Manager Cue() method returns with a value of 0, each sub-filter (IBMAudioMT, IBMVideoMT, MuxMT, and FileStoreMT) will have already created a thread that will perform the encoding for that filter. Each of those four threads waits to begin encoding until the filter's Start() method releases a semaphore.

long Start(). Actually starts the encoding process by releasing a semaphore and starting the encoding thread in each of the sub-components. A return of 0 indicates only that the semaphore was released successfully, not that the encoding process has finished (or even begun successfully). To monitor the progress of the encode, you might call GetCurrentFrameCount(), comparing its return value with that of GetDurationFrameCount(). Otherwise, you can wait for a finished event to determine when the encoding process has finished. See the Events section, below.

long Stop() – Stops the encode. Actually, this is one of three methods of stopping an encode:

- Prior to the start of each encode, an encode duration is set in the mux Registry table for the current board. If neither Stop() nor End() is called, this duration parameter controls the automatic stopping of the encoder
- Calling the Stop() command causes the encoding process to stop immediately, or abort, without flushing buffers.
- Calling the End() command causes the encoding process to stop cleanly, after fully processing the frame that is currently in the mux component and flushing it out to file.

A return of 0 from the Stop() or End() method indicates only that Filter

Manager received the call. You must wait for a finished event to determine when the encoding process has actually halted. See the Events section, below.

long End() – This is the alternate and preferred method of stopping an encoding session. It performs a clean stop, guaranteeing that the frame currently being processed by the mux component will be flushed through the storage modules before the encode halts. See the discussion of stopping encodes, above. A return of 0 from the End() method indicates only that the Filter Manager received the call. You must wait for a finished event to determine when the encoding process has actually halted. See the Events section, below.

long Pause() – Causes the encoding process to pause immediately. Note that with each set of pause / resume calls, the audio-video synchronization may be affected slightly. We recommend no more than 3 or 4 pauses per encode. A return of 0 from the Pause() method indicates only that the Filter Manager received the call. You must wait for a paused event to determine when the encoding process has actually paused.

long Resume() – Causes the encoding process to resume immediately after a pause.

long Reset() – Causes all of the encoder's COM components to reset themselves in preparation for the next encode. This command should be issued prior to the Load() / Cue() pair if an error occurred during the previous encode.

long Destroy() – Called when exiting your application, this method closes handles, frees memory, and otherwise cleans up the multi-board filter manager component.

Events

The multi-board Filter Manager uses the COM event mechanism to send messages to its client through the outgoing interface. An *event* is similar to a callback issued by the Filter Manager in response to a noteworthy occurrence or condition. Any client of the Filter Manager may elect to register for and respond to Filter Manager events. Most likely, you'll want your client to register to receive them. After you have called the Start() method, only through events can the Filter Manager let your application know that it has finished encoding, and whether or not it completed successfully.

From the Filter Manager's outgoing interface, a message and a code are passed each time an event is fired. The message is used to display a detailed message string describing the event. In the case of an error event, the code specifies the error that occurred. For events other than error events, the code parameter may be 0, indicating a successful outcome. Codes with negative values are generally error codes, a list of which can be found in Appendix B. All others are status codes.

The multi-board Filter Manager supports these events:

HRESULT ErrorEvent(long code, BSTR message) – Issued when a processing error has occurred. When your application receives an error event, it should set a flag indicating that an error has occurred. The Cue() method should check this error flag, calling Reset() if the flag is set.

HRESULT LogEvent(long code, BSTR message) – Issued to inform the client of the status of the encoding process or of a recently issued command. Informational only. If the code value is negative, the log event can be considered a warning.

HRESULT FinishedEvent(long code, BSTR message) – Issued to inform the client that the current encoding session has finished.

For an explanation of how to register events in your C++ or Visual Basic application, see the respective sections of this manual that describe C++ and Visual Basic client applications. For a more detailed explanation of COM events, refer to the COM references listed in Chapter 1, particularly the book *ATL Internals*.

Summary

Using the methods and properties of the primary interface, combined with the three methods of the events interface, you will be able to create an application that controls the multi-board Argus encoder. For each board that you wish to manage, just follow these steps:

Setting up the multi-board Filter Manager object:

1. Create a MultiBoardFilterMgr object (let's refer to its Smart Pointer as pFM) and register for events issued by that object.

2. Call pFM->PutEncoderBoardNumber(x) to indicate that this instance of the Filter Manager will control board x.

3. Call pFM->Initialize() to initialize the Filter Manager. Ascertain that the return value of the method is 0 (not a negative return code) before continuing.

Preparing for an encode (Repeat steps 4—8 for each stream to encode):

4. If an error condition was flagged during the previous encode, call pFM->Reset(). Check the result before continuing.

5. Programmatically modify any Windows Registry values that must be reset for

this particular encoding session. (The source code provided with the MBProps application demonstrates the setting of Windows Registry values using the CRegistry class.)

6. Call pFM->Load() to load the encoding parameters from the Windows Registry. If the return value of the Load() call is less than 0, abort the encode, flag an error condition, and notify the user that the load has failed.

7. Call pFM->Cue() to cue all of the filters in preparation for the upcoming encoding session. If the return value of the Cue() call is less than 0, abort the encode, flag an error condition, and notify the user that the cue has failed. If the cue succeeds, you might want to call Save() to save the current encoder parameters to the Registry. Note that the Filter Manager, through its Load() and Cue() commands, may adjust a few of the Windows Registry settings if it detects inconsistencies.

8. When you are ready to start the encode, call pFM->Start(). The Start() command activates the encoding threads of all of the filters by releasing a semaphore for each, causing the encode to begin two frames later. The Start() command returns immediately. If the encoding process finishes successfully, the Filter Manager will fire a finished event, to which your application should respond by resetting its state in preparation for the next encode. If an error occurs before the encode finishes, the Filter Manager will fire an error event. In response, your application should flag an error condition, triggering a call to Reset() on the next cue (see step 4). Notify the user that the encode has aborted or finished successfully.

9. Repeat steps 4 through 8 for each encoding session.

Before exiting the application:

10. Call the events EasyUnadvise() method, then delete the Events object.

11. Call pFM->Destroy() to free memory and close handles.

12. Call pFM.Release() to release the Filter Manager.

13. Call CoUninitialize() to close the COM library and free COM resources on the current thread.

Sample Applications

Overview

Developers using the multi-board Filter Manager (MultiBoardFilterMgr) should be familiar with Microsoft Visual C++ 6.0 and/or Microsoft Visual Basic. Microsoft provides several wizards and tools that make adding COM support to your C++ or Basic applications relatively straightforward. While it is possible to access and use these components from other development environments, only examples for Visual C++ are provided in this SDK. Other packages with full COM support should behave similarly.

The general steps for setting up a client application are:

1. Create the client project.
2. Initialize the COM libraries (VC++ only)
3. Create an instance of the desired object. (In VC++, use the Smart Pointer to the interface).
4. Use the object.
5. Release the object (C++ only)
6. Uninitialize the COM libraries when finished.

The remaining sections of this section describe and explain three working example applications that control various aspects of the Argus encoding process. When the Argus SDK is installed, the source code for each of these applications can be found in C:\Program Files\ Vela Research \Argus\SDK. The intent of providing the source code for each of these applications is to illustrate the use of various programming tools to control some aspect of the encoding process. In order to present readable, easy-to-follow code, we have intentionally kept the applications simple.

The "FourBoardTestApp" Sample Application

The Sample Visual C++ Multi-Board Encoder-Control Application

Overview

A full set of source code for a C++ interface to the multi-board Argus encoder is provided with this SDK. The application (including source code, the Filter Manager type library, and Registry-based DLLs) is located in C:\Program Files\Vela Research\Argus \ SDK\FourBoardTestApp. It would be useful to refer to a copy of the source code as you read the section that follows. Most of the source code referenced in this document is located in FourBoardTestApp-Calls.cpp and FilterManagerEvents.cpp.

Creating the Project

When you are creating a Microsoft Foundation Class (MFC) Application Wizard EXE project like MBTestApp, it is very important that you select from the App Wizard window the check box that adds support for ActiveX controls. This inserts into the StdAfx.h file the header files required to support the COM libraries. If you are adding COM support to an existing project, or if your project does not use MFC, we highly suggest that you study some of the books available on MFC core details and COM specifics. We also suggest that you use MFC as a shared DLL from App Wizard, as all of the COM components are already using this DLL.

Initializing the COM Libraries

When you create a client of a multi-board Argus COM filter, you must first initialize the COM libraries by calling CoInitializeEx(), the COM initialization method that fully supports multi-threaded programming. The code to initialize the COM libraries looks like this:

```
{
    HRESULT hr = CoInitializeEx( NULL, COINIT_MULTITHREADED );
    if ( FAILED(hr) )
            return FALSE;
}
```

CoInitializeEx() must be called in each thread of your application before any other COM-related calls are made. Each call to CoInitializeEx() must be matched with a call to CoUninitialize().

All of the multi-board Argus core components support a dual-interface. A dual interface is any interface that inherits from IDispatch, which is the interface that supports OLE Automation. Using the custom interface of the component provides the client with direct table access to the functionality of the component. This is much more efficient than the IDispatch interface, which uses the COM Automation libraries to access component functionality.

In general, if you are using version 6.0 of Visual C++, it is best to use the custom interface directly. The IDispatch interface is provided for backward compatibility with Visual Basic 4.0 and earlier development, as well as to access the components from scripting languages such as VBScript and JScript.

Using the #import Directive

This section describes the steps required to create a COM object using Smart

Pointers and the **#import** compiler directive. The #import directive is a Microsoft-specific compiler directive that creates a Smart Pointer wrapper class from a type library. That class can be used to create an instance of the required COM object and to use its services. To use the #import directive for an instance of the multi-board Filter Manager interface, you would insert the following code in the StdAfx.h file after the #include <afxwin.h> directive.

```
#include <afxwin.h>             // MFC core and standard components
#include <afxext.h>             // MFC extensions
#include <afxdisp.h>            // MFC Automation classes
#include <afxdtctl.h>           // For IE4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // Windows Common
#endif                          // _AFX_NO_AFXCMN_SUPPORT

#include <atlbase.h>
extern CComModule _Module;
#include <atlcom.h>
#include <objbase.h>
#import "MultiBoardFilterMgr.tlb"  no_namespace named_guids
```

The CComModule class implements a COM server, allowing the client to access the module's components. When you open your application, you should call _Module.Init(NULL, AfxGetInstanceHandle()). When you close the application, call _Module.Term(). If you fail to do this, many of the multi-board Filter Manager features, including events, will not work properly. For an example, please refer to the InitInstance() method in FourBoardTestApp.cpp.

The #import directive creates two header files that reconstruct the type library contents in C++ source code. In this case, the files would be named MultiBoardFilterMgr.tlh and MultiBoardFilterMgr.tli. The primary header file, MultiBoardFilterMgr.tlh, contains a typedef macro that expands to the following format:

```
typedef
com_ptr_t<com_IIID<IMBFilterMgr,__uuidof(IMBFilterMgr)>>IMBFilterMgrPtr
```

The C++ template class _com_ptr_t, used in the above typedef, creates a Smart Pointer (in this case, IMBFilterMgrPtr) that can be used to access the interface passed in as the template argument (in this case, IMBFilterMgr).

Creating an Instance of the Multi-Board Filter Manager Interface

In order to use the commands and properties that the multi-board Filter Manager exposes, you'll need to create an instance of the Filter Manager interface, then register to receive Filter Manager events. The Create() method defined in FourBoardTestAppCalls.cpp shows the source code that is required to create instances of the primary and the event interfaces of the MultiBoardFilterMgr component. Note that the Create() method establishes a Filter Manager object that will control the first encoder board. To control the second, third and fourth boards, the application defines the Create1(), Create2(), and Create3() methods.

The Smart Pointer interface defined in MultiBoardFilterMgr.tlh makes it easy to create an instance of the multi-board Filter Manager. You simply need to call CreateInstance(). Note that the only argument to the CreateInstance() method is the class ID of the multi-board Filter Manager component. Because we specify the **named_guids** modifier, the #import directive creates the required CLSID in the header file for us, eliminating the need to call CLSIDFromProgID.

```
m_pIMBFilterMgrPtr.CreateInstance(CLSID_MBFilterMgr);
```

Once an instance of IMBFilterMgr has been created, it can be used to access the properties and methods exposed through the primary multi-board Filter Manager interface.

For a more thorough discussion of Smart Pointers, please refer to *ATL Internals*, by Rector and Sells.

Setting Up an Event Sink Object

Now that you have created an IMBFilterMgr object to control the first encoder board, you'll need to set up your application to receive log, error, and finished events from the multi-board Filter Manager COM component for that same board. Setting up your application to receive events involves creating an event sink object, then connecting to the Filter Manager event source. Later in this chapter, in the section entitled "Registering to Receive Multi-Board Filter Manager Events," we'll discuss the details of registering to receive events. For now, we'll just briefly mention that there are two methods that should be called immediately after the multi-board Filter Manager CreateInstance() call, described above. These two methods are:

```
m_pFilterManagerEvents = new CFilterManagerEvents();
m_pFilterManagerEvents->EasyAdvise(m_pIFilterMgr);
```

The first method creates the event sink object. The second advises the multi-board Filter Manager event interface that we are connecting to it. The EasyAdvise() method, which you will write yourself, is described in detail in the "Registering to Receive Multi-Board Filter Manager Events" section.

Using the Object

Once the IMBFilterMgrPtr is created, we can use that pointer to call any of the methods that the interface makes available. The following segment of code, for example, is the method that the sample application calls to set up or “cue” the first encoder board in preparation for an encode. There are similar methods (OnCue1(), OnCue2(), OnCue3()) that are used to cue the remaining three encoder boards. Within the OnCue() method, the actual calls to the multi-board Filter Manager interface are Load(), Cue() and GetDurationFrameCount().

```
void CMBTestAppDlg::OnCue()
{
    long lRetval = 0;
    if( EncoderState == esError )
    {
            lRetVal = m_pIFilterMgr->Reset();
            EncoderState = (lRetval < 0) ? esError : esInitialized;
    }
    SetButtons();
    lRetVal = m_pIFilterMgr->Load();
    if( lRetval < 0 )
    {
            CString msg;
            msg.Format(
                    _T(“Error Loading Encode Parameters: %ld.”),
                    lRetval);
            MessageBox(msg);
            return;
    }
    lRetval = m_pIFilterMgr->Cue();
    if( lRetval < 0 )
    {
            CString msg;
            msg.Format(_T(“Error on Cue: %ld”), lRetval );
            MessageBox(msg);
            return;
    }
    long lFrames = m_pIFilterMgr->GetDurationFrameCount();
    m_nRequestedFrames = lFrames;
```

```
            UpdateData(FALSE);
            EncoderState = esCued;
            SetButtons();
        }
```

Note that we always check the return value of Filter Manager methods to ensure that the method succeeded. COM-related errors raise exceptions, while the components themselves return a long result, which is typically set to 0 if successful or to a negative error code if not. For a complete list of Filter Manager error codes, refer to Appendix B.

Releasing the COM Libraries

Like all resources, the COM libraries must be released when the program is finished using them. For this purpose, COM provides a single method, CoUninitialize(), which should be invoked in the destructor of the client code.

Before calling CoUninitialize(), first release the Filter Manager interface by calling:

```
        m_pIFilterMgr.Release();
        m_pIFilterMgr = NULL;
```

Registering to Receive Filter Manager Events

When you're programming in C++, registering to receive events is not the easiest of tasks. However, ATL does provide a template class, IDispEventImpl, to assist C++ client applications in receiving events from the server.

Before registering to receive events, make certain that you have already called _Module.Init() to initialize the data members of the COM server module. This should be done when you initialize your application. Also remember to call _Module.Term() before exiting your application.

These are the steps that you'll need to follow to register for events using the **IDispEventImpl** template:

1. **Derive a class from the IDispEventImpl template.** For example, in our sample application, we derive CFilterManagerEvents from IDispEventImpl (see FilterManagerEvents.h):

```
        class CFilterManagerEvents :
            public IDispEventImpl<1, CFilterManagerEvents>
```

The first argument to IDispEventImpl is a unique identifier for the event source. Since the Filter Manager is the only source from which our sample application receives events, its ID of 1 is, indeed, unique.

2. **In the class definition, insert a SINK_MAP that includes each of the events from MultiBoardFilterMgr that you'd like to receive.** For example, in

our sample application, we register to receive the error, log, finished, and pause events.

```
BEGIN_SINK_MAP(CFilterManagerEvents)
    SINK_ENTRY(1, 1, OnError)
    SINK_ENTRY(1, 2, OnLog)
    SINK_ENTRY(1, 3, OnFinished)
    SINK_ENTRY(1, 4, OnPause)
END_SINK_MAP()
```

The arguments for SINK_ENTRY are (SOURCE, DISPID, FUNC), where:

- SOURCE identifies the event source. Since the Filter Manager was identified by a value of '1' in step 1, above, we use the value '1' here to indicate that we're receiving events from the Filter Manager.
- DISPID identifies the dispatch ID within the event source of the event that we're receiving. In the Filter Manager, the error event has a dispatch ID of 1, the log event has a dispatch ID of 2, the finished event has a dispatch ID of 3, and the pause event has a dispid of 4.
- FUNC identifies the function or method that will handle the received event. This method will be defined within the class that we're currently deriving from the IDispEvent Impl.

3. **Define and implement a method that calls** AtlGetObjectSourceInterface() **and** DispEventAdvise()**.** First call AtlGetObjectSourceInterface() to retrieve pUnk, a pointer to the interface ID of the default source interface. Then call DispEventAdvise() to establish a connection with the event source represented by pUnk. This connection allows events fired from pUnk (or, in our case, from the Filter Manager) to be routed to the handler functions specified in our event sink map.

In our C++ sample application, the EasyAdvise() method is included in the CFilterManagerEvents class to perform the connections described in the paragraph above:

```
HRESULT EasyAdvise(IUnknown* pUnk)
{
    // The COM object corresponding to pUnk must
    // implement IProvideClassInfo2 or IPersist*.
    // Call this method to extract info about the
    // source type library if you specified only 2
    // parameters to IDispEventImpl
    HRESULT hr = AtlGetObjectSourceInterface(
                    pUnk, &m_libid, &m_iid, &m_wMajorVerNum,
                    &m_wMinorVerNum);

        // connect the sink and source
        hr = DispEventAdvise(pUnk, &m_iid);
        return hr;
    }
```

4. **Within the CFilterManagerEvents class, define and implement a method that calls *AtlGetObjectSourceInterface()* and *DispEventUnadvise()*.** Once again, AtlGetObjectSourceInterface() is called to retrieve pUnk, a pointer to the event source. DispEventUnadvise() breaks the connection with the event source represented by pUnk. Once the connection is broken, events will no longer be routed to the handler functions.

In our C++ application, the EasyUnadvise() method is included in the CFilter-ManagerEvents class:

```
    HRESULT EasyUnadvise(IUnknown *pUnk)
    {
        AtlGetObjectSourceInterface( punk, &m_libid, &m_iid,
                    &m_wMajorVerNum, m_wMinorVerNum);
        return DispEventUnadvise(punk, &m_iid);
    }
```

5. **Within the CFilterManagerEvents class, define and implement the functions that will handle the events that you've registered for.** Within the body of each of these event handlers, insert code to respond in whatever way you decide to the event that you're receiving. For example, you may choose to write out to a log file any messages that you receive from a log event.

Within the class definition of our sample application, the event handlers are prototyped as follows:

```
    STDMETHOD(OnError)(long code, BSTR error);
    STDMETHOD(OnLog)(long code, BSTR error);
    STDMETHOD(OnFinished)(long code, BSTR error);
    STDMETHOD(OnPause)(long code, BSTR error);
```

Following is an example of the implementation of the log event handler, copied from FilterManagerEvents.cpp. Note that we call MessageBox() for demonstration purposes only. For deliverable applications, never hold up a log event with a function requiring user input.

```
STDMETHODIMP
CFilterManagerEvents::OnLog(long code, BSTR error)
{
   CString strMessage = _T("");

   // Displaying this message in a message box is
   // for demo purposes only. DO NOT use Message
   // Boxes (or any other method requiring
   // extensive processing or requiring user input)
   // in production code event handlers! Doing so
   // may lock up the encoder.
   if( error )
   {
            BSTR localBstr = error;
            int strLen = ::SysStringLen(localBstr);
            if( strLen > 0 )
            {
                     CComBSTR tempBstr(strLen, localBstr);
                     strMessage = tempBstr;
                     if( ( code < 0 ) && pView )
                     {
                              pView->MessageBox( strMessage );
                     }
            }
   }
   if( !pFile )
            return S_OK;
   // Send error message to log file.
   if( code < 0 )
            _ftprintf(  pFile,
                     _T("Warning %ld: %s\n"),
                     code, strMessage );
   else
            _ftprintf( pFile, _T("%s\n"), strMessage);
   fflush( pFile );
   return S_OK;
}
```

When you implement the event interfaces, it is important to consider what Argus software threads or processes will be firing them. Most events generated by the Argus API will be called in the context of their own thread, which is engaged in near real-time processing with the encoder hardware. If event interface methods cause excessive delays, exceptions to be raised, or the calling thread to wait indefinitely on a synchronization object, undesirable behavior is sure to result. If these types of operations must be performed in response to an event, create your own thread, return control to the calling thread, and proceed to do the processing in the context of your own thread.

For example, if receiving a finished event should trigger your application to start another encode, make certain that your implementation of the finished event just toggles a flag or a semaphore, which prompts another thread (preferably the main thread) to cue the next encode. This will ensure that the threads invoking the event interfaces remain responsive to hardware events.

It is important, too, to realize that by implementing an event interface, you have effectively made your client code a server for those events. This means you need to take into account the synchronization of data accessed by objects using the event interface.

Running the Sample Application

The FourBoardTestApp sample C++ application, portions of which have been discussed in the previous sections of this chapter, is intended as an example of the use of the multi-board Filter Manager interface, not as an end-user application. However, it is useful to compile and run the sample application to observe and understand the operation of the Argus encoder.

Before running the sample application, first visit "MultiBoard FilterMgr Interface Properties," page 21, to set up the encoding parameters for each installed encoder board. Next, make certain that you have a running audio/video source connected to each encoder board. Then run FourBoardTestApp.

When the FourBoardTestApp main dialog first pops up, click on the Initialize button. In performing its initialization, the MultiBoardFilterMgr component creates a Filter Manager object for each installed board. If the initialization procedure succeeds for each board, the dialog will change so that only the Cue button for each board is active. If the initialization fails, all buttons for the board in question will remain inactive.

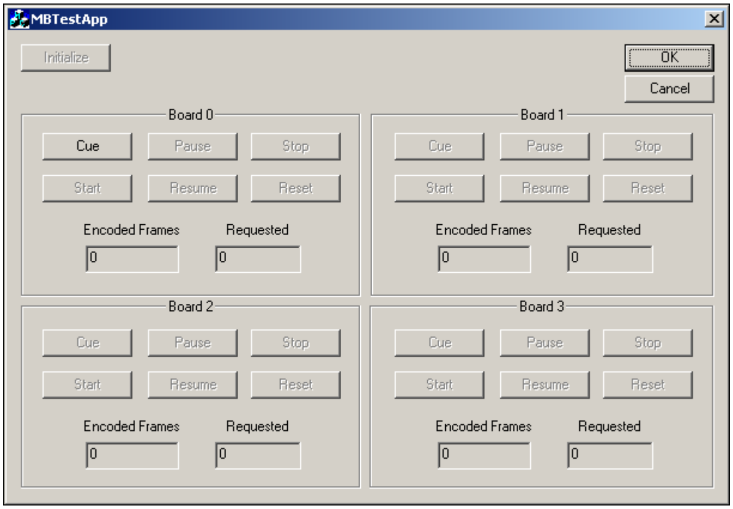
Now click on the Cue button to start an encode for one of the boards. The Filter Manager Cue() method resets all of the sub-components, loads the encoding parameters from the Registry, then cues each of the sub-components in preparation for the start of an encode. If all of these procedures complete successfully, the FourBoardTestApp dialog will change so that only the Start, Stop, and Reset buttons for the encoder board are active. Additionally, the requested number of video frames will appear in the Requested text window.

When you (and your source) are ready to begin encoding, click Start. Note that the Encoded Frames text field is updated each second with the number of frames already encoded. The encode will end when the requested number of frames have been encoded. At that point, the application will receive a finished event from the Filter Manager, and the dialog will change once again so that only the Cue button is active for the board that just finished encoding.

During the encode, you can click on the Pause and Resume buttons to pause and resume the current encode. However, if you pause the encode too many times, you risk losing audio/video synchronization.

If an error should occur sometime during the encode, the sample application will receive an error event. In response, it writes a message to the log file, then calls the Filter Manager Stop() and Reset() methods.

Note that you should be able to simultaneously to run multiple encoding sessions, one for each installed board.



Encoded Frames: 0 Requested: 0 (each of Board 0, Board 1, Board 2, Board 3)

Figure 2-3. C++ Sample Application Window

MBProps

Sample Visual C++ Registry-Control Application for Multiple-Board Encoding

Overview

Most of the properties that must be defined before the start of an encode are set in Argus-specific Windows Registry tables. Many of these properties can be set once when the encoder software is installed, then can be left unchanged thereafter. Specific applications, however, may need to reset a subset of the encoder properties before each encode. For example, most applications will assign a new MPEG file path name with each encode. Other applications may allow the user to change the video bit rate or resolution with each encode.

In order to adjust specific Windows Registry settings before cueing for an encode, you may need to control the Registry programmatically. To allow programmatic access to the encoder Registry tables, Vela has designed a C++ class (CRegistry), the source code to which is provided in the MBProps sample application. MBProps uses the CRegistry class to read from and to write to the encoder Registry tables. We encourage you to use the CRegistry class to access and modify the encoder Registry tables. Example screen shots developed from the MBProps program are shown at the end of this section.

If you have programmed using our standard Argus SDK, you'll notice that MBProps is similar to the RegCtrlPnl application. The biggest difference is that each Registry table managed by MBProps (and subsequently used by the multi-board Argus encoder) has a single digit appended to its name. For example, the table that holds the mux properties for the first encoder board is named Mux0. The digit appended to the table name indicates the encoder board to which the properties apply, where the number of the first board is '0.' Each of the encoder-specific multi-board Registry tables is listed and described in Appendix A.

CRegistry Methods

The CRegistry class provides the following five methods (or, in the case of numbers 3 and 4, types of methods).

1. **Constructor:** Creates an instance of the class and initializes its members.
2. **Open:** Opens the Registry. Returns TRUE if successful, FALSE otherwise.
3. **SetValue:** Writes a setting to the Registry. There are a number of SetValue() methods defined in the CRegistry class, each of which handles a specific data type. The SetValue() method is usually called with two arguments: the name of the Registry key and the value to be saved to that Registry key. The method returns TRUE if it is successful.

4. **GetValue:** Reads a setting from the Registry. Again, there are a number of GetValue() methods defined in the CRegistry class, each of which handles a specific data type. The GetValue() method is usually called with three arguments: the name of the Registry key, a pointer to a variable to hold the value read, and a default value to be given to the variable if no Registry setting is available. The method returns TRUE if it is successful.
5. **Destructor:** Closes the Registry table.

Example: Loading an Encoder Registry Table

As an example of using the CRegistry class to load settings, the following excerpt of MBProps source code loads the file-store settings from the “MuxStore0” Registry table. Again, this code is provided as an example of the calls required to read values from the Registry. In a production-quality application, you would be likely to add more error-checking.

```
//////////////////////////////////////////////////////////
// InitializeMuxStoreSettings() reads the settings for the // mux-store property
page from the Registry, setting the // windows controls accordingly.
//////////////////////////////////////////////////////////
void MuxStore::InitializeMuxStoreSettings()
{
    CRegistrySettings;
    CRegistryMuxStore;
    unsigned long val;

    // m_strTableName = “MuxStore0”;

    if( Settings.Open(HKEY_CURRENT_USER, ARGUS_KEY) == TRUE )
    {
                if( MuxStore.Open(Settings.hKey(),
                            m_strTableName) == TRUE )
                {
                            MuxStore.GetValue( T(“LocalFilename”),&m_strFileName,
                                        _T(“D:\\MpegFiles\\Test.mpg”));
                            MuxStore.GetValue(_T(“OptimizedMuxWrites”), &val,
                                        TRUE);
                            m_bMuxStoreOptimize = (val != 0);
```

```
                        MuxStore.GetValue( _T(“StorageEnabled”), &val,
                                FALSE);
                        m_bMuxStoreEnabled = (val != 0);
                        MuxStore.Close();
                }
                Settings.Close();
        }

        // Remainder of code for this function pertains to
        // dialog settings and has been omitted.
    }
```

The above code segment begins by opening the HKEY_CURRENT_USER\Software\Vela Research\Broadcast Argus branch of the Registry with the Settings.Open() call. Note that HKEY_CURRENT_USER is defined in winreg.h, and ARGUS_KEY is defined in CRegistry.hpp. We chose to store Argus properties in HKEY_CURRENT_USER so that we could programmatically both read and write settings to the Registry, even when the logged-in user does not have administrative privileges. HKEY_LOCAL_MACHINE is read-only for any users that are not set up with administrative privileges.

If the Broadcast Argus branch of the Registry is opened successfully, the method calls MuxStore.Open() to open the MuxStore0 table. Finally, it calls MuxStore.-GetValue() to retrieve the value stored in the Registry for the “LocalFilename,” “OptimizedMux Writes,” and “StorageEnabled” keys. As you can see, the first argument to GetValue() is the Unicode-compliant name of the key whose value is being read. The second argument is the address of the variable whose setting is being read from the Registry. The third and last GetValue() argument is the default value that the variable should take if the key cannot be found or read from the Registry.

When the keys in the table are read successfully, Close() is called on all CRegistry objects that were successfully opened.

Example: Storing Values in an Encoder Registry Table

As an example of using the CRegistry class to store settings, the following excerpt of MBProps source code saves file-store settings in the “MuxStore0” Registry table. Note that all of the Open() and Close() calls are the same as in the InitializeMuxStoreSettings() method above. The only difference is that, instead of calling GetValue() to read from the Registry, the following code calls SetValue() to write to the Registry. The SetValue() method takes just two arguments: the

Unicode-compliant name of the Registry key that is being set and the value to which that key is being set.

```
////////////////////////////////////////////////////////////////////
// SaveMuxStoreSettings() reads the settings of the mux-store property
// page controls, then stores those values to the Registry.
////////////////////////////////////////////////////////////////////
void MuxStore::SaveMuxStoreSettings()
{
    CRegistrySettings;
    CRegistryMuxStore;

    UpdateData(true);
    m_bMuxStoreOptimize = m_ctlMuxOptimize.GetCheck() == 1;
    m_bMuxStoreEnabled = m_cltMuxStoreEnabled.GetCheck() == 1;

    if( Settings.Open(HKEY_CURRENT_USER, ARGUS_KEY) == TRUE )
    {
            if( MuxStore.Open(Settings.hKey(),m_strTableName) == TRUE )
            {
                    MuxStore.SetValue(_T(“LocalFilename”), m_strFileName);
                    MuxStore.SetValue(_T(“OptimizedMuxWrites”),
                            m_bMuxStoreOptimize);
                    MuxStore.SetValue(_T(“StorageEnabled”),
                            m_bMuxStoreEnabled);
                    MuxStore.Close();
            }
            Settings.Close();
    }
}
```

For More Information on Registry Control

For more examples of the use of the CRegistry class, review the source code for the MBProps application. In addition to defining CRegistry, this application has a .cpp file to manage each Registry table that holds encoder properties. Each MPProps property page allows you to specify the encoder board whose properties you are reading and writing. The application opens only Registry tables whose names are appended with the specified board number.

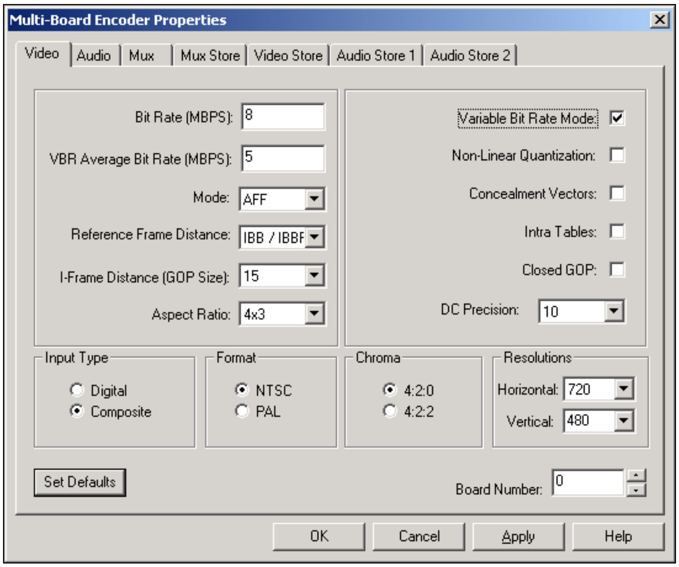
For information regarding the multi-board Registry settings for each of the table types, please refer to Appendix A.

MBProps Typical Screen Shots

When you double-click on MBProps.exe, a tabbed property sheet appears. By default, the application begins by displaying the encode properties for encoder board 0. Each page of the property sheet displays the encode properties that are stored in one of the multi-board Registry tables described in Appendix A.

Whenever you change a property on one of the property pages, the Apply button becomes active. Just click on Apply to write to the Registry the changed property values. To restore the current property page to its factory settings, click the Set Default button. To change the board whose properties you are viewing or modifying, just click on Select Board, enter the board number, and click on Apply. The settings on all of the property pages will change to correspond to the board number that you requested.

Sample screen shots of the property pages appear on the following pages.



Multi-Board Encoder Properties

Video | Audio | Mux | Mux Store | Video Store | Audio Store 1 | Audio Store 2

Bit Rate (MBPS): 8
VBR Average Bit Rate (MBPS): 5
Mode: AFF
Reference Frame Distance: IBB / IBBF
I-Frame Distance (GOP Size): 15
Aspect Ratio: 4x3

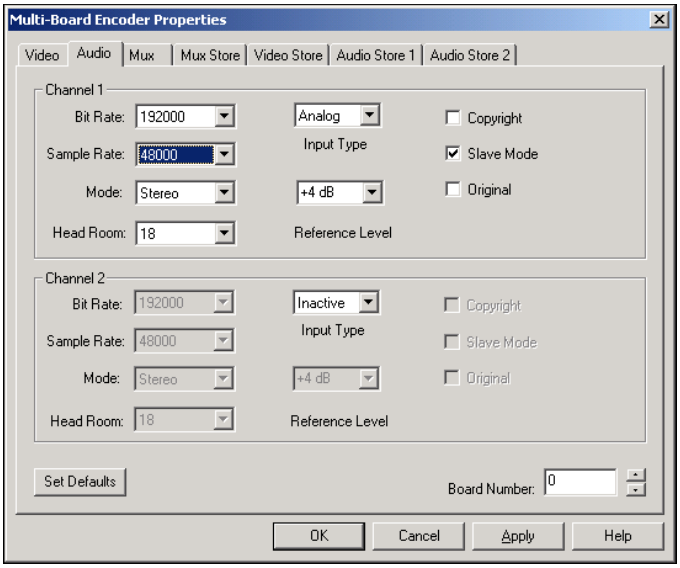
Variable Bit Rate Mode: ✓
Non-Linear Quantization: ☐
Concealment Vectors: ☐
Intra Tables: ☐
Closed GOP: ☐
DC Precision: 10

Input Type: Digital [✓ Composite]
Format: [✓ NTSC] PAL
Chroma: [✓ 4:2:0] 4:2:2
Resolutions: Horizontal: 720 Vertical: 480

Set Defaults
Board Number: 0

OK Cancel Apply Help

Figure 2-4. MBProps — IBM Video Properties



Channel 1

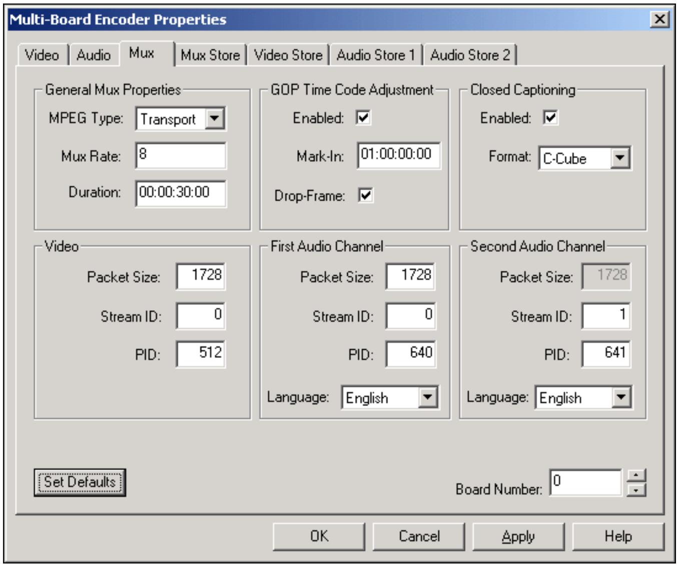
Bit Rate: 192000
Sample Rate: 48000
Mode: Stereo
Head Room: 18
Input Type: Analog
Reference Level: +4 dB
[ ] Copyright
[✓] Slave Mode
[ ] Original

Channel 2

Bit Rate: 192000
Sample Rate: 48000
Mode: Stereo
Head Room: 18
Input Type: Inactive
Reference Level: +4 dB
[ ] Copyright
[ ] Slave Mode
[ ] Original

Board Number: 0

Figure 2-5. MBProps — IBM Audio Properties



General Mux Properties

MPEG Type: Transport
Mux Rate: 8
Duration: 00:00:30:00

GOP Time Code Adjustment

Enabled: [✓]
Mark-In: 01:00:00:00
Drop-Frame: [✓]

Closed Captioning

Enabled: [✓]
Format: C-Cube

Video

Packet Size: 1728
Stream ID: 0
PID: 512

First Audio Channel

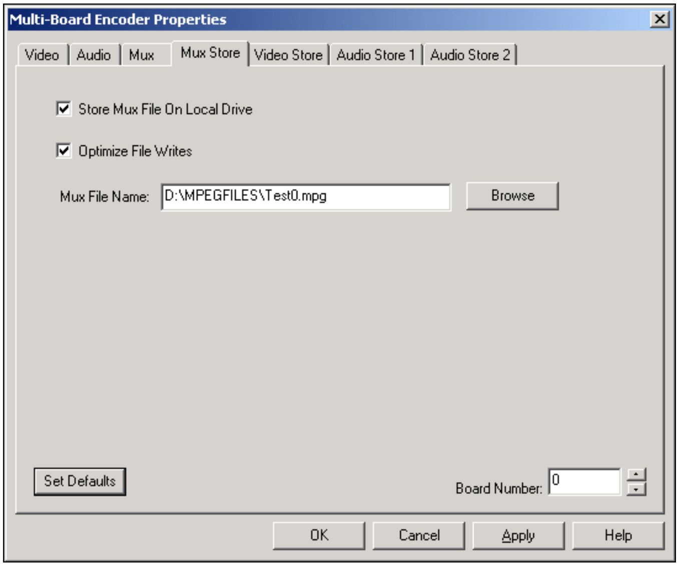
Packet Size: 1728
Stream ID: 0
PID: 640
Language: English

Second Audio Channel

Packet Size: 1728
Stream ID: 1
PID: 641
Language: English

Board Number: 0

Figure 2-6. MBProps — Mux Properties



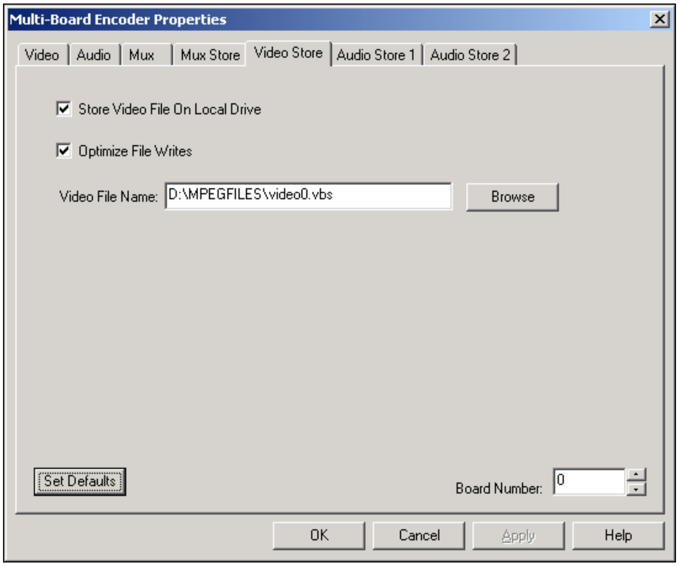
☑ Store Mux File On Local Drive

☑ Optimize File Writes

Mux File Name: D:\MPEGFILES\Test0.mpg

Board Number: 0

Figure 2-7. MBProps — Mux Store Properties



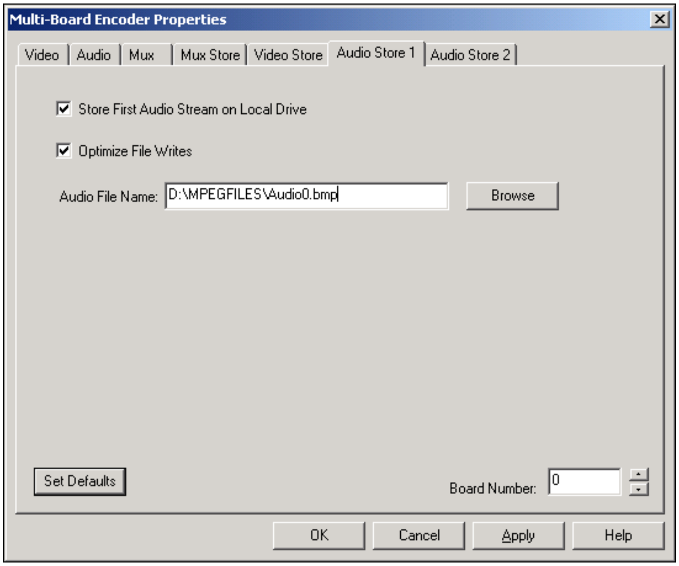
☑ Store Video File On Local Drive

☑ Optimize File Writes

Video File Name: D:\MPEGFILES\video0.vbs

Board Number: 0

Figure 2-8. MBProps — Video Store Properties



☑ Store First Audio Stream on Local Drive

☑ Optimize File Writes

Audio File Name: D:\MPEGFILES\Audio0.bmp

Board Number: 0

Figure 2-9. MBProps — Audio Store Properties

Distributing Components

Overview

Building an installation disk is very important. This is the first view a user will have of your software system. It is crucial that the installation procedure install your software easily and correctly. The last thing you want to hear from a user is that the software won't install.

For the multi-board Argus encoder we use Wise version 8.1 to build our software installation package. The resulting installation disks are robust and easy to follow, and they give a nice presentation to the package. All of the necessary steps required by our components (driver and component registration, Windows Registry setup, etc.) are handled automatically by this software. If this product meets your needs, use it. If not, there are many alternative installation products.

The concept of this section is to give you some direction on creating your installation disks. It is very important that certain features of the component architecture be installed correctly in order for the encoder to function properly. You will, of course, be required to add the portions that you have created (any *.exe files and required *.dll files) to the install script. We list the files that are needed for your install script, and where they can be found on an installed multi-board Argus system. It is important that these files be placed into the same directory structure on the destination machine.

The following issues must be addressed within the installation procedure in order for the board(s) and components to function correctly:

- Driver installation / Registry settings
- Redistributable files
- Microcode directory structure
- Multi-board Argus COM components and registration

Multi-Board Installation Using Single-Board Driver

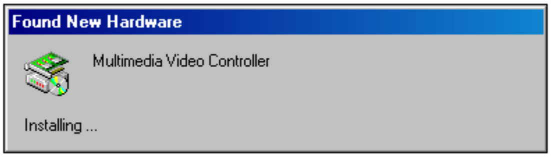
The multi-board Argus requires the Windows 2000 WDM Vela single-board encoder device driver **velasbe.sys**. Only users with Administrator privileges can install hardware drivers. The Vela single-board encoder device driver uses the Plug and Play installation methods. First, install the hardware. Then use either the Windows Hardware Wizard or the Device Manager to install and/or associate the correct driver for each encoder board as follows.

When you install a new device, Windows 2000 will detect and configure it. For Plug and Play PCI devices, such as the Vela single-board encoder, just shutdown, power off, and install the board(s). When you restart the system, log in as a user with Administrator privileges. Windows 2000 enumerates the device and starts the Plug and Play installation procedure automatically.

1. You first see a pair of “Found New Hardware” windows (Figure 2-10). The small window identifies the hardware based on what Windows 2000 knows about the device. In this case, because the Vela single-board encoder's PCI configuration data identifies it as a Multimedia Video Controller, the board is initially identified as a “Multimedia Video Controller.”

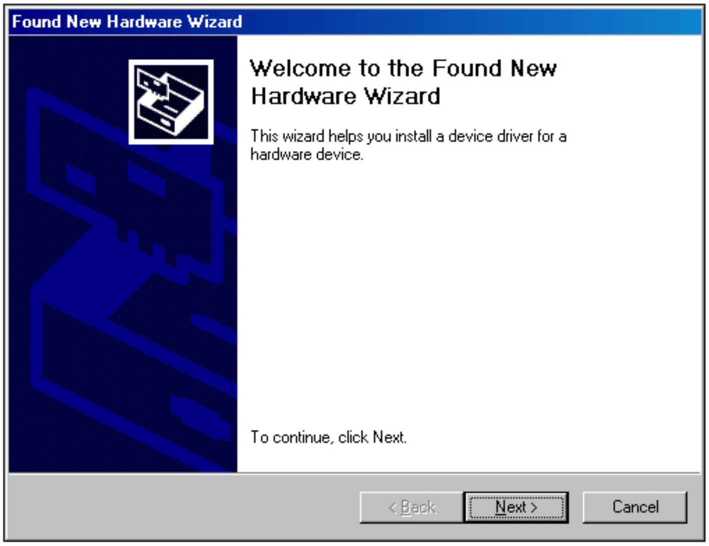
2. When you click “Next,” the “Install Hardware Device Drivers” window opens (Figure 2-11). Select “Search for a suitable driver for my device,” and click Next.

3. The “Locate Driver Files” window opens (Figure 2-12). Check “CD-ROM drives” and uncheck any other “Optional search locations”. Click Next.


Found New Hardware

Multimedia Video Controller

Installing ...

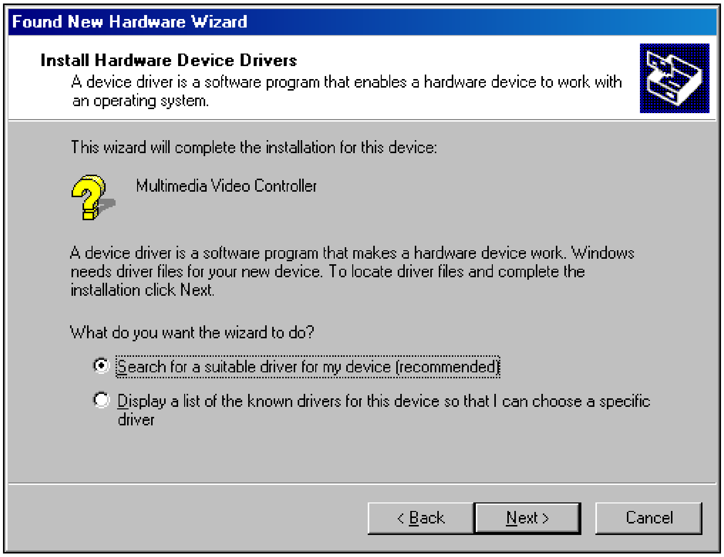

Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install a device driver for a hardware device.

To continue, click Next.

Figure 2-10. "Found New Hardware" Windows



Found New Hardware Wizard

Install Hardware Device Drivers
A device driver is a software program that enables a hardware device to work with an operating system.

This wizard will complete the installation for this device:

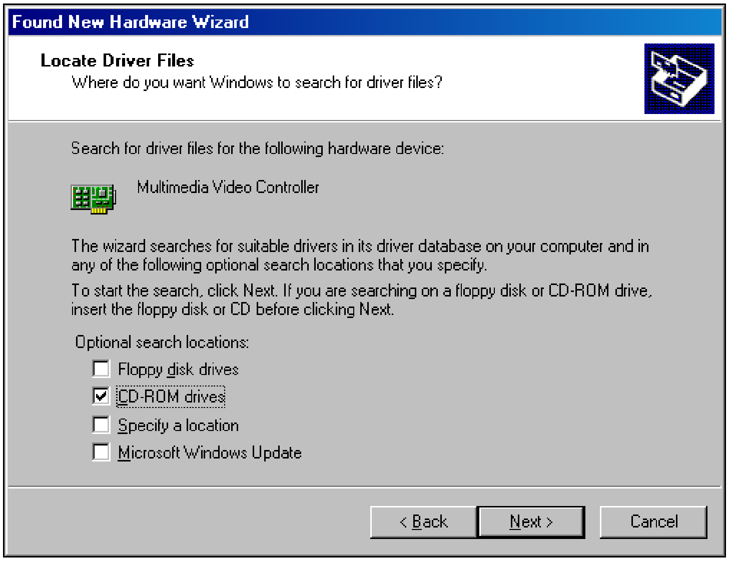
Multimedia Video Controller

A device driver is a software program that makes a hardware device work. Windows needs driver files for your new device. To locate driver files and complete the installation click Next.

What do you want the wizard to do?

- [✓ Search for a suitable driver for my device (recommended)]
- Display a list of the known drivers for this device so that I can choose a specific driver

Figure 2-11. “Install Hardware Device Driver” Window



Found New Hardware Wizard

Locate Driver Files
Where do you want Windows to search for driver files?

Search for driver files for the following hardware device:

Multimedia Video Controller

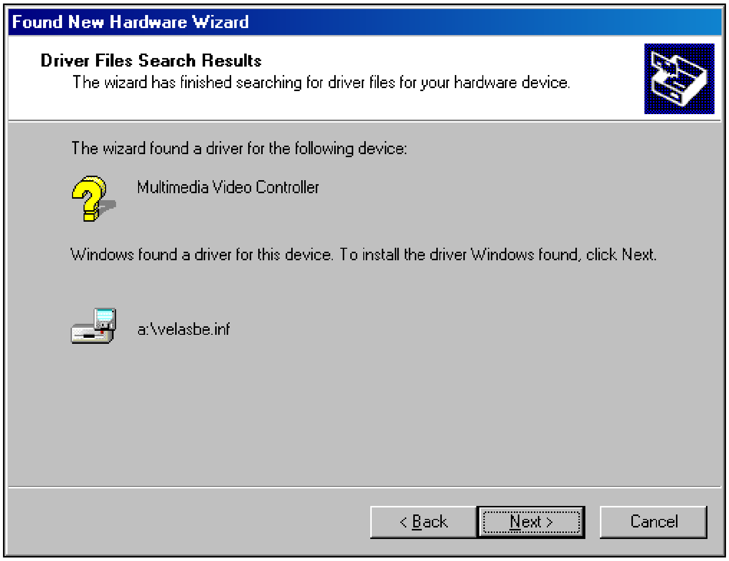
The wizard searches for suitable drivers in its driver database on your computer and in any of the following optional search locations that you specify.

To start the search, click Next. If you are searching on a floppy disk or CD-ROM drive, insert the floppy disk or CD before clicking Next.

Optional search locations:

- ☐ Floppy disk drives
- ☑ CD-ROM drives
- ☐ Specify a location
- ☐ Microsoft Windows Update

Figure 2-12. “Locate Driver Files” Window


Found New Hardware Wizard

Driver Files Search Results

The wizard has finished searching for driver files for your hardware device.

The wizard found a driver for the following device:

Multimedia Video Controller

Windows found a driver for this device. To install the driver Windows found, click Next.

a:\velasbe.inf

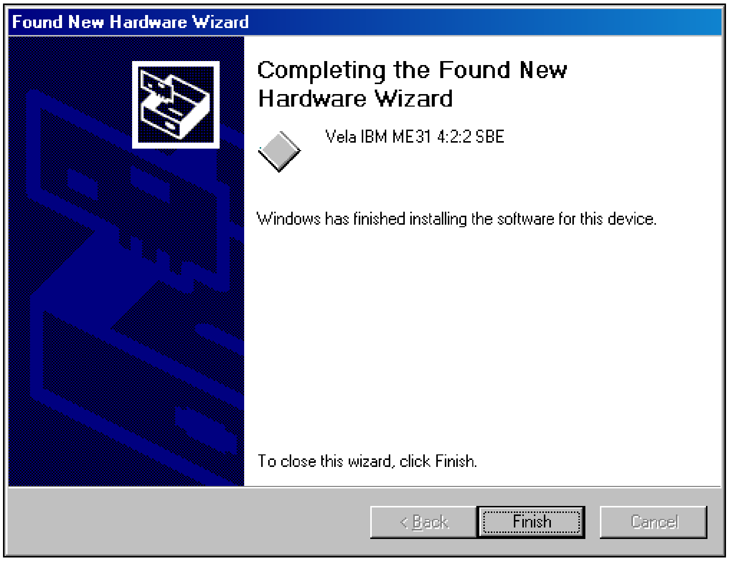
Figure 2-13. "Driver Files Search Results" Window

4. The "Driver Files Search Results" window (Figure 2-13) appears when an INF file on the CD-ROM is located that contains a match for the device identification.

   After you click Next here, the setup process will interpret the INF file, establish the Registry settings to identify this hardware as a Vela Single Board Encoder family board, and copy the driver **velasbe.sys** and INF file velasbe.inf to the appropriate system directory locations. From this point on, the system will be able to identify the boards by the friendly names specified in the velasbe.inf file.

   A number of windows briefly appear as the setup process copies the **velasbe.sys** driver and velasbe.inf files as directed by the INF file.

   When the driver installation completes, the "Completing the Found New Hardware Wizard" window (Figure 2-14) appears.


Found New Hardware Wizard

Completing the Found New Hardware Wizard

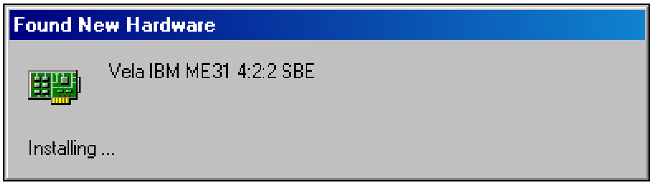
Vela IBM ME31 4:2:2 SBE

Windows has finished installing the software for this device.

To close this wizard, click Finish.

Figure 2-14. "Completing Found New Hardware Wizard" Window

5. The board (in this example, the 3-chip 4:2:2 SBE) is now identified correctly. Click Finish to proceed. If there are more SBE's to install at this point, Windows Plug and Play setup will continue with the installation of the remaining boards.


Found New Hardware

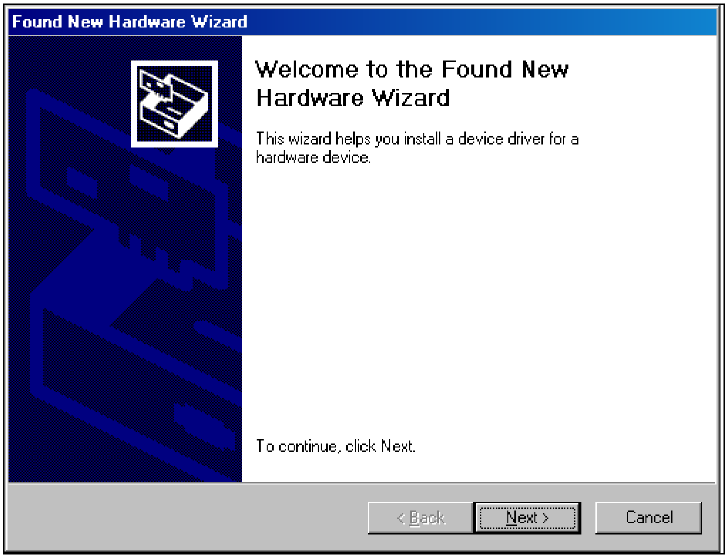
Vela IBM ME31 4:2:2 SBE

Installing ...

6. First you will see the "Found New Hardware" pop-up window, shown above, this time identifying the second single-board encoder by its registered friendly name.

7. The "Found New Hardware Wizard" window will also open (Figure 2-15). Click Next.

8. The "Install Hardware Device Drivers" window (Figure 2-16) opens. Select "Search for a suitable driver for my device" and click Next.



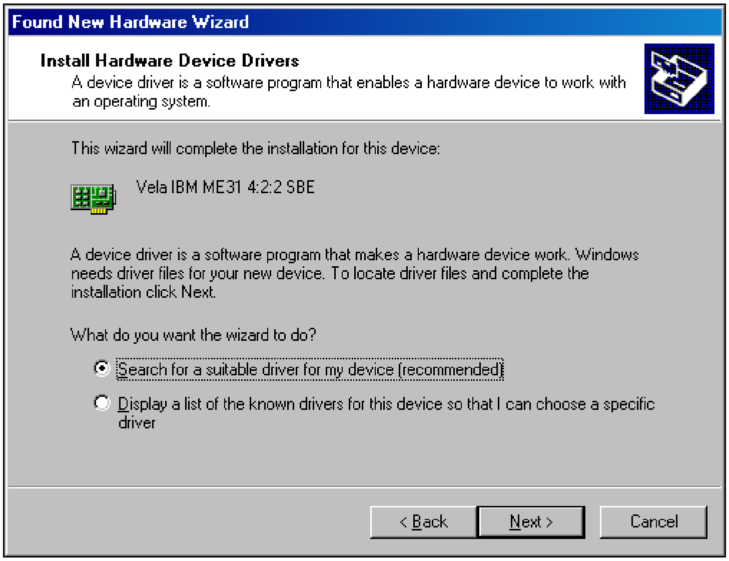
Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install a device driver for a hardware device.

To continue, click Next.

Figure 2-15. "Found New Hardware Wizard" Window



Found New Hardware Wizard

Install Hardware Device Drivers

A device driver is a software program that enables a hardware device to work with an operating system.

This wizard will complete the installation for this device:

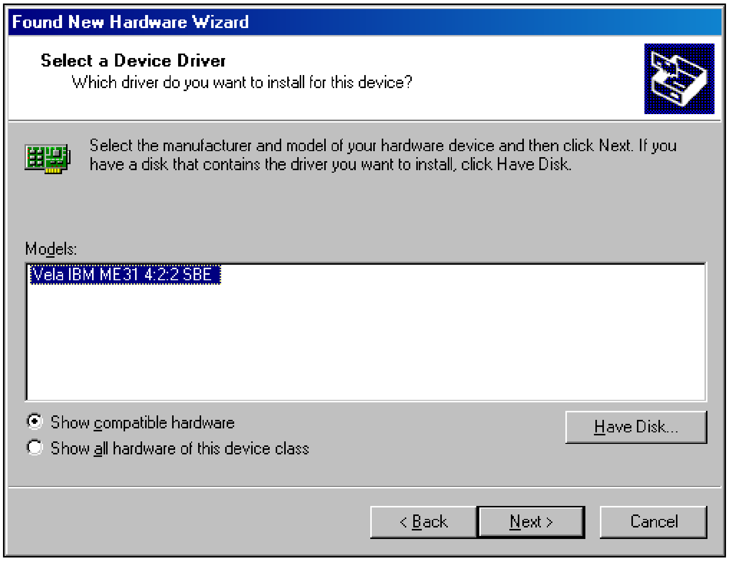
Vela IBM ME31 4:2:2 SBE

A device driver is a software program that makes a hardware device work. Windows needs driver files for your new device. To locate driver files and complete the installation click Next.

What do you want the wizard to do?

- [✓ Search for a suitable driver for my device (recommended)]
- Display a list of the known drivers for this device so that I can choose a specific driver

Figure 2-16. "Install Hardware Device Drivers" Window


Found New Hardware Wizard

Select a Device Driver
Which driver do you want to install for this device?

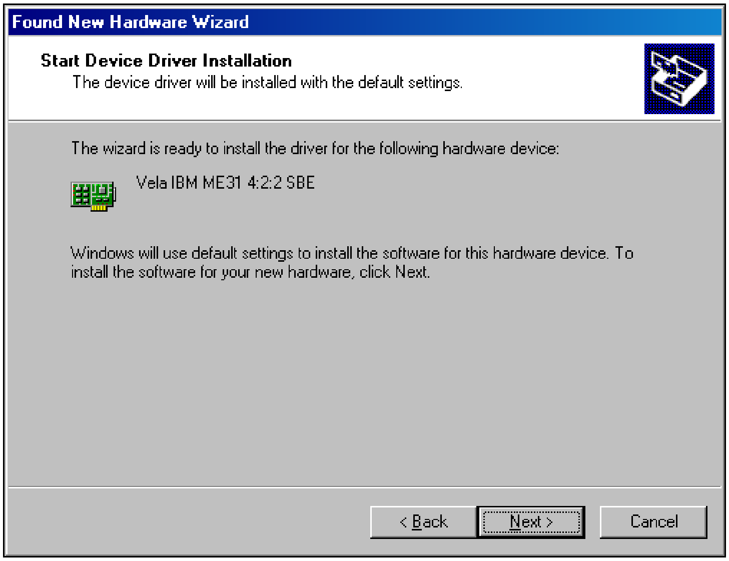
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.

Models:
Vela IBM ME31 4:2:2 SBE

[✓ Show compatible hardware]
Show all hardware of this device class

Figure 2-17. "Select a Device Driver" Window

9. (Figure 2-17) The system Registry now contains the necessary information to identify the device and its driver from the previous installation process. The "Vela IBM ME31 4:2:2 SBE" device is automatically selected. Click Next.


Found New Hardware Wizard

Start Device Driver Installation

The device driver will be installed with the default settings.

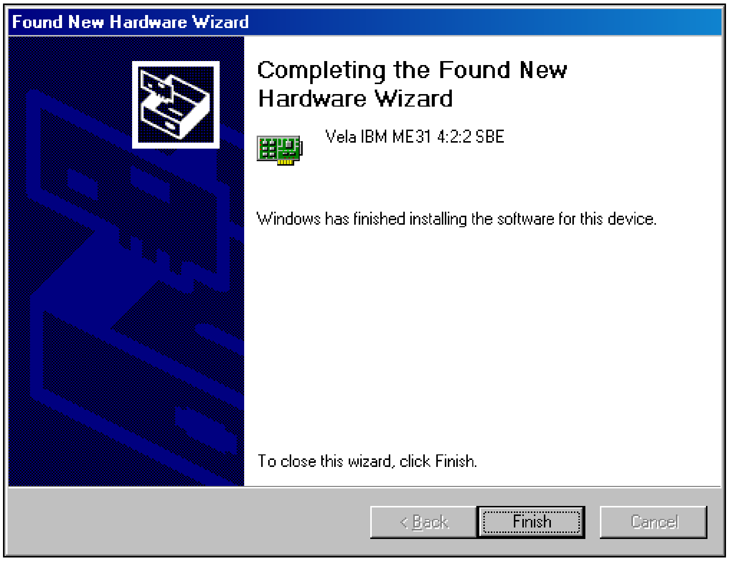
The wizard is ready to install the driver for the following hardware device:

Vela IBM ME31 4:2:2 SBE

Windows will use default settings to install the software for this hardware device. To install the software for your new hardware, click Next.

Figure 2-18. "Select a Device Driver" Window

10. The "Start Device Driver Installation" window appears (Figure 2-18). If the installation media is still present, the files will be copied again when you click Next. Otherwise, you will be asked to locate the installation files, as before.


Found New Hardware Wizard

Completing the Found New Hardware Wizard

Vela IBM ME31 4:2:2 SBE

Windows has finished installing the software for this device.

To close this wizard, click Finish.

Figure 2-19. "Completing Found New Hardware Wizard" Window

11. The "Completing the Found New Hardware Wizard" window appears (Figure 2-19). Click Finish.

After the last board is installed, you can verify that the driver installed and the board is functioning correctly by using the System Properties / Device Manager control panel.

12. Select "My Computer" on the desktop, right click and select "Properties." The System Properties control panel will open. Then click Device Manager.

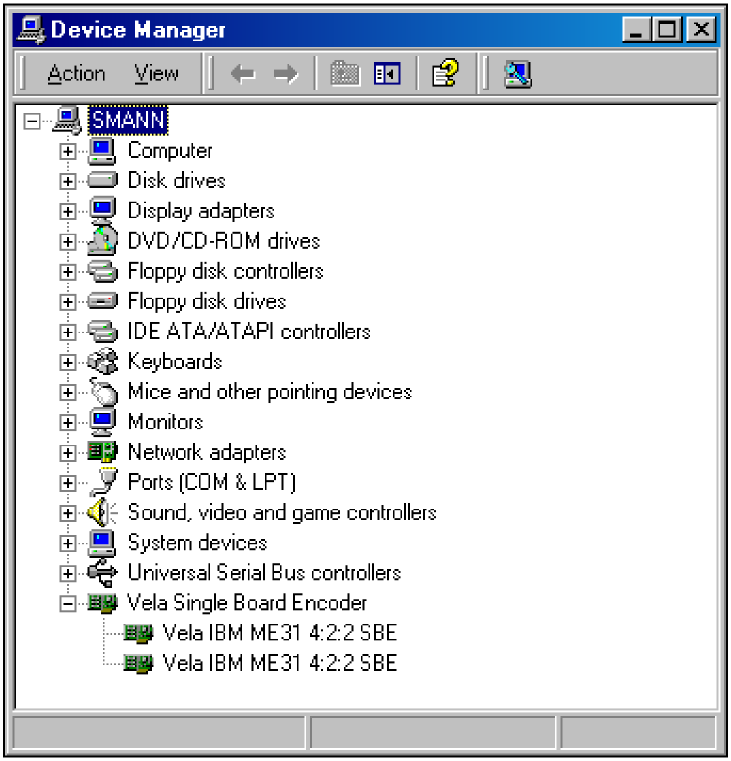


Figure 2-20. Device Manager Window

The Device Manager window appears (Figure 2-20). The velasbe.inf INF file declares a device class with the name Vela Single Board Encoder and three different hardware devices that belong to the class:

- Vela IBM ME31 4:2:2 SBE
- Vela IBM S420 SBE
- Vela IBM S422 SBE

Device Manager displays a hierarchical view of device class/devices. In this example, we installed two Vela IBM ME31 4:2:2 SBE boards.

When the driver installs correctly and the devices are functioning correctly and have no resource conflicts, the display appears as shown the illustration.

Microsoft Redistributable Code

The current installation requires two sets of Microsoft redistributable code:

- MFC Class Libraries:
  - mfc42.dll
  - mfc42u.dll
  - msvcrt.dll
  - wininet.dll
- COM Registration:
  - atl.dll
  - comctl32.dll
  - comctl32.ocx
  - olepro32.dll
  - regsvr32.exe

File names may differ depending on the version used. For more information, see the online help for Microsoft developers. This list can also be found on the Microsoft online help files for Distributing ActiveX Controls. In addition to the core registration files, the file atl.dll must be added to allow the COM objects to self-register.

Microcode Directory Structure

The IBM encoder microcode is loaded onto the encoder hardware at encode time. It is installed on Argus encoders in two folders: C:\etc\422 for the 3-chip encoder boards, and C:\etc\S422 for the 1-chip boards. It must be placed in the correct sub-folder of C:\etc in order to load.

The microcode files used by the multi-board encoder are:

- Set 1: ME31 V1-FAST Studio Profile Set (3-chip board):
  - c:\etc\422\h4_03.bin
  - c:\etc\422\i4_03.bin
  - c:\etc\422\f4_1e.bin
  - c:\etc\422\r2_19.bin
- Set 2: S-Series Microcode (S422 and S420 boards):
  - c:\etc\S422\i5_10.bin
  - c:\etc\S422\e5_10.bin
  - c:\etc\S422\r3_10.bin
  - c:\etc\S422\md5_10.bin

The last two digits of the microcode file represent the version number of the binary file. As IBM releases new microcode revisions for the ME31, S422, and S420 encoder chipsets, the multi-board Argus software will incorporate them.

COM Components

The following COM libraries, which comprise the API set, are distributed with the multi-board Argus application. These COM objects are registered automatically with each installation of the standard product.

- Argus COM Components Located in C:\Program Files\Argus:

  MultiBoardFilterMgrU.dll

  FileStoreMT.dll

  IBMAudioMT.dll

  IBMVideoMT.dll

  MultiplexMT.dll
- Argus COM Components Located in C:\Program Files\Common:

  MemMgrMT.dll

  MemMgrServerMT.dll

  PinsMT.dll
- Optional COM Component Registered by SDK Installation:

  Vtr.dll

Component Registration

In order for multi-board Argus software to run, all of the COM components listed above must be registered. The multi-board Argus installation procedure registers all of the standard COM components. If the components are installed on a system without an automatic registration program like Wise, you can register them using the **regsvr32.exe** application provided with the multi-board Argus software.
To register a COM component, type:

```
Regsvr32 /s <COM component filename>
```

Regsvr32 is a redistributable utility that Microsoft provides at no extra charge.

Error Codes

Each of the methods exposed through the main interface of the MBFilterManager COM component returns a value, typically a long. A return value of 0 indicates that the method completed without error. However, a negative return value indicates that an error was encountered.

Similarly, once an encoding session has begun, you may receive a negative value for the first argument of an error, finished, or log event fired by the Filter Manager interface. Once again, a negative code indicates that an error has occurred.

Appendix B contains a table with all possible Argus Filter Manager error codes.

Customer Support

In the event of questions or problems with Vela Application Programming Interface methods, materials, or this manual, do not hesitate to contact Vela Training and Support as follows:

- Phone: (727) 507-5301
- E-mail: support@vela.com
- World Wide Web - http://www.vela.com

Chapter 3

Using the VTR API

Component Overview

Lois, please check this out for applicability re the multi-board.

The multi-board API does not currently support VTR control. If you wish to control a VTR as part of the encoding process, you might want to use the VTR API, detailed in this chapter, to monitor the position of the tape in preparation for starting the encode. As a result of this change, the duration of the encode is now set in the Mux table.

If you want to control a tape deck programmatically while not actively encoding, the Vela Software Developer's Kit provides a stand-alone VTR component that uses Sony 9-pin protocol to communicate through a serial port with a tape deck. (References on the 9-pin protocol are available on the Internet or can be ordered from Sony.) This particular COM component is called VTR.dll. Its type library, VTR.tlb, is inserted in the C:\Program Files\Vela Research\Argus\SDK \TypeLibs folder when the SDK is installed. The customer interface provided for the VTR component is IVTRCenter.

Note that VTR.dll should not be confused with VTRControl.dll, a component of the Argus encoder software architecture that is directly managed by Filter Manager. If the "Source Enabled" key in the "VTR" Windows Registry is set to 1, VTRControl.dll is responsible for controlling the VTR during the encoding process, starting with the cue phase of the encode. Minimally, if the "Source Enabled" key is set to 0, VTRControl.dll is responsible for logging and managing the duration of the encode.

On the other hand, the VTR component is an "extra." You can successfully run the encoder without using it. Though it is not required, the VTR component provides you with a set of methods to control the tape deck **between** encodes. For example, you can use it to tell the tape deck to fast-forward, to rewind, to jog or shuttle. Or, in preparation for setting your mark-in or mark-out value, you can instruct the VTR component to retrieve the current time code from the tape deck. Alternatively, you could substitute your own VTR-control software or a third-party package.

The center panel of "FMTestApp," the C++ sample encoder application, offers a set of buttons and edit fields that illustrate the use of a few of the methods of the VTR component. Also included with the Software Developer's Kit is "VTRTestApp," an application that more comprehensively illustrates the use of the methods exposed through the VTR COM component. Source code is provided.

If the Filter Manager component already has control of the serial port used to control the tape deck, the commands issued by the VTR component will not work.

Therefore, the Filter Manager component exposes a VTRDisconnect() method. The Filter Manager VTRDisconnect() method allows you to detach from the serial port before giving control of the port to the VTR component. Likewise, the Filter Manager VTRConnect() method can be used to return control of the serial port to the Filter Manager component prior to an encode. Of course, these two Filter Manager methods are meaningful and effective only when the “VTR” Windows Registry “Source Enabled” key is set to 1.

Similar to Filter Manager, the VTR component has a Connect() and a Disconnect() method. These methods allow the VTR component to assume or to relinquish control of the encoder serial port attached to the tape deck. Before calling any of the commands exposed through the VTR interface, you should first call the Connect() method. Before returning control of the serial port to the Filter Manager component, you should call the Disconnect() method. Make certain not to attempt to assume control of the tape deck while an encode is cueing or in progress.

Windows Registry Settings

The VTR component uses a single Windows Registry value to set the serial port delay (the minimum amount of time, in milliseconds, to wait after sending a command to the tape deck via the serial communications port). The Windows Registry key used to define this setting is:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Vela Research\Argus\SerPortDly.
```

If this Registry key is not defined, the VTR component uses a default value of 10 (translated as 10 milliseconds).

NOTE: Do not call VTRDisconnect() from the error-event or finished-event handler.

Creating an Instance of IVTRCenter

You can access the VTR COM component through a single custom interface, IVTRCenter. To generate a Smart Pointer to this interface, first call (in the StdAfx.h file):

```
#include <atlbase.h> // ATL Support
#import “VTR.tlb” no_namespace named_guids
```

In your class definition, you should define a Smart Pointer to the VTR interface:

```
IVTRCenterPtr m_IVtr;
```

Then, in the initialization section of your application, create an instance of the interface:

```
m_IVtr.CreateInstance( CLSID_VTRCenter );
```

Immediately after creating an instance of the VTR component, you should set and initialize the communications port, as follows:

```
m_IVtr->PutComPort(1);
long result = m_IVtr->Initialize();
```

If the value returned by Initialize() is 0, then the serial-port initialization was successful, and you're ready to use the methods and properties exposed through the IVTRCenter interface.

Properties Exposed Through IVTRCenter

The following properties are exposed through the IVTRCenter interface:

ComPort – This property, a long, identifies which of the communication ports on the encoder is connected to the tape deck. On most Vela encoders, the number of each of the communication ports is marked on the chassis. Typically the ComPort property will be set to either 1 or 2.

The ComPort property can be retrieved by calling val = GetComPort(). It can be set by calling PutComPort(val). In both cases, the variable "val" is defined as a long.

DropFrame – This property, a BOOL, reports whether or not the current time code read from the tape deck is a drop frame time code. To retrieve the value of this property, call val = GetDropFrame(), where val is defined as a BOOL. If the GetDropFrame method returns 1, the time code is drop-frame time code. If the method returns 0, the time code is a non-drop-frame time code. There is no Put() method available for this property.

HardError – This property, a BOOL, determines whether or not the VTR interface encountered an error during the previous operation or method call. Use the call val = GetHardError() to retrieve the value of the property. If the value returned is 1, an error was encountered during the last operation. If the return value is 0, the last operation was successful.

MarkIn – This value, a BSTR, represents the current value of the inpoint or mark-in set on the tape deck. The format of the property is "hh:mm:ss:ff", where "01:02:03:04" represents a mark-in of 1 hour, 2 minutes, 3 seconds, and 4 frames.

There are two access methods, Get() and Put(), available for this property. The Get() method queries the tape deck to determine the current setting of its mark-in. It returns the retrieved value to the calling application. The Put() method accepts a mark-in value as an argument, sending a command to the tape deck to set its inpoint or mark-in to that value. Where val is defined as a **_bstr_t**, the Get() and Put() methods for MarkIn are called as follows:

```
val = m_IVtr->GetMarkIn();
m_IVtr->PutMarkIn(val);
```

MarkOut. This value, a BSTR, represents the current value of the outpoint or mark-out set on the tape deck. The format of the property is "hh:mm:ss:ff", where "01:02:03:04" represents a time code of 1 hour, 2 minutes, 3 seconds, and 4 frames.

There are two access methods, Get() and Put(), available for this property. The Get() method queries the tape deck to determine the current setting of its out-point. It returns the retrieved value to the calling application. The Put() method accepts time code as an argument, sending a command to the tape deck to set its out-point or mark-in to that value.

Where val is defined as a **_bstr_t**, the Get() and Put() methods for MarkOut are called as follows:

```
val = m_IVtr->GetMarkOut();
m_IVtr->PutMarkOut(val);
```

TapeInserted. This property, a BOOL, determines whether or not there is a tape inserted in the tape deck. The single access method available for this property is GetTapeInserted(). It returns a value of 1 if there is a tape inserted, 0 if there is not.

TimeStamp. This property, a BSTR, represents the time stamp currently on the tape deck. Its format is "hh:mm:ss:ff." For example, a time stamp of "01:02:03:04" translates as 1 hour, 2 minutes, 3 seconds, and 4 frames. There is a single access method available for this property. Where val is defined as a **_bstr_t**, the access method is:

```
val = m_IVtr->GetTimeStamp();
```

Calling GetTimeStamp() prompts the VTR component to query the tape deck for the current time code, which the Get() method then returns.

VTRRemoteMode. This property, a BOOL, determines whether the encoder is set to local or remote mode. There is a single access method available for VTRRemoteMode. Where val is defined as a BOOL, the access method is:

```
val = m_IVtr->GetVTRRemoteMode();
```

Calling the Get() method prompts the VTR component to query the tape deck as to whether it is in remote or local mode. The method returns 1 if the tape deck is set to remote-control mode, or to 0 if the deck is set to local-control mode.

VTRType. This property, a long, indicates the type of protocol used to communicate with the tape deck. Both Get() and Put() methods are available to access VTRType. The Get() method returns a long, and the Put() method accepts a long as an argument.

Currently the only supported value for this property is SONY9_PIN(0).

Methods Exposed Through IVTRCenter

The methods exposed through the IVTRCenter interface fall into three categories: those used to initialize the component, those used to manage the serial port, and those used to control the tape deck itself.

Component Initialization Method

long Initialize() – This method should be called immediately after a Smart Pointer to the IVTRCenter interface is created. It connects to and sets up the serial port in preparation for sending commands to or receiving information from the tape deck. Note that you should identify the serial port used for tape-deck communications **before** calling Initialize(). Do this by calling PutComPort(val).

The Initialize() method returns 0 if successful, 1 if not.

Serial Communications Port Management Methods

long Connect()
long Disconnect() – As discussed at the beginning of this chapter, these methods allow you to connect or to disconnect from the serial port used to communicate with the tape deck. Before returning control of the tape deck back to the Filter Manager, you should call Disconnect() on the IVTRCenter interface. To return control of the tape deck to the VTR component, first call the Filter Manager VTRDisconnect() method, then call the IVTRCenter Connect() method.

Both of these methods return 0 on success, -1 on error.

Tape Deck Control Methods

The remaining methods of the IVTRCenter interface allow you to issue commands to the tape deck using Sony 9-pin protocol. The tape-deck control methods available through the IVTRCenter interface are described below.

long Eject() – Issues a command to the tape deck to eject the tape.

Returns:
 0 if the command was successful
-9 if the command failed
-32 if VTRType is not set to SONY_9_PIN.

long FFwd() – Issues a command to the tape deck to fast-forward the tape.

Returns:
0 if the command was successful
-17 if the command failed
-32 if VTRType is not set to SONY_9_PIN.

long GetPreRoll(long *pPreRoll) – Issues a command to the tape deck to retrieve the number of seconds to pre-roll. The pre-roll (in integral seconds) is returned at the address passed as the only argument to GetPreRoll().

The method returns:
0 if the command was successful
-19 if the command failed
-32 if VTRType is not set to SONY_9_PIN.

long GotoPreRoll() – Issues a command to roll the tape to `<pre-roll>` seconds before the mark-in, where both the mark-in and the pre-roll have been set by earlier Set() or Put() commands.

The method returns:
0 if the command was successful
-33 if the command failed
-32 if the VTRType is not set to SONY_9_PIN.

long GotoTimeCode(BSTR TimeCode) – Issues a command to roll the tape to the time code indicated by the TimeCode argument, which is expressed as a string in the format "hh:mm:ss:ff."

The method returns:
0 if the command was successful
-58 if the command failed
-32 if the VTRType is not set to SONY_9_PIN.

long Jog(long direction, long speed) – If direction is set to 1, jogs the tape forward. If the direction is set to -1, jogs the tape backward. The speed can assume any integer value from 0 to 9, defined as followed:

0 – Still
1 – 1/100 of play speed
2 – 1/10 of play speed

5 – Play speed
6 – 2.9 times play speed
9 – 5 times play speed.

The method returns:
0 if the command was successful
-51 if the command failed
-33 if the VTRType is not set to SONY_9_PIN.

long Pause() – Issues a command to pause the tape deck.

The method returns:
0 if the command was successful
-14 if the command failed
-33 if the VTRType is not set to SONY_9_PIN.

long Play() – Issues a command to play the tape deck.

The method returns:
0 if the command was successful
-4 if the command failed
-33 if the VTRType is not set to SONY_9_PIN.

long Rwnd() – Issues a command to rewind the tape deck.

The method returns:
0 if the command was successful
-11 if the command failed
-33 if the VTRType is not set to SONY_9_PIN.

long SetPreRoll(long Preroll) – Issues a command to set the pre-roll to the value passed in as an argument, which is expressed in integral seconds.

The method returns:
0 if the command was successful
-44 if the command failed
-33 if the VTRType is not set to SONY_9_PIN.

long Shuttle(long ShuttleSpeed) – This command operates the same as the Jog command, except that it takes just one argument. If ShuttleSpeed is a value greater than 0, the tape deck shuttles forward at the speed indicated in the table below. If the ShuttleSpeed is a negative value, the tape deck shuttles backward. Use the absolute value of ShuttleSpeed to determine the actual shuttle speed, as follows:

0 – Still
1– 1/100 of play speed

2 – 1/10 of play speed
5 – Play speed
6 – 2.9 times play speed
9 – 5 times play speed.

The method returns:
0 if the command was successful
-33 if the VTRType is not set to SONY_9_PIN
-41 if the shuttle speed is an invalid value
-42 if the command failed for any other reason.

long Stop() – Issues a command to stop the tape deck.

The method returns:
0 if the command was successful
-7 if the command failed
-33 if the VTRType is not set to SONY_9_PIN.

Appendix A

Multi-Board Encoder Registry Settings

Overview

The multi-board Argus encoder reads all of its encoding properties from the Windows Registry. For each encoder board installed in the system, there is a set of seven Registry tables from which the application reads all of its encoding properties and to which it stores the set of encoding properties used during the most recent encode performed by that encoder board. Each of the seven Registry table names ends in a single digit, specifying the number of the encoder board to which the properties apply. The multi-board Argus Registry tables are listed below. The 'X' at the end of each represents the encoder board number, where the first board number is 0.

```
\HKEY_CURRENT_USER\Software\Vela Research\Broadcast Argus\IBM AudioX
\HKEY_CURRENT_USER\Software\Vela Research\Broadcast Argus\IBM VideoX
\HKEY_CURRENT_USER\Software\Vela Research\Broadcast Argus\MuxX
\HKEY_CURRENT_USER\Software\Vela Research\Broadcast Argus\MuxStoreX
\HKEY_CURRENT_USER\Software\Vela Research\Broadcast Argus\VideoStoreX
\HKEY_CURRENT_USER\Software\Vela Research\Broadcast Argus\FirstAudStoreX
\HKEY_CURRENT_USER\Software\Vela Research\Broadcast Argus\SecondAudStoreX
```

If the multi-board Argus Registry locations listed above are not established prior to the first encode, a call to the Filter Manager Load() method will create the tables, providing default settings for each of the keys. These settings can be modified in one of three ways:

- Programmatically. See the discussion of the CRegistry class at “CRegistry Methods,” page 39.
- Manually, using regedt32 or regedit.
- Using MBProps (see “MBProps,” page 39.)

The multi-board Filter Manager interface exposes two special methods that load all of the encoder properties from and save all of the encoder properties to the Registry:

long Load() – Loads into memory all of the Registry settings for the current encoder board. If the Registry location does not exist, the Load() call creates it, creates all of the Registry keys, then assigns to each key its default value. You should call the Load() method prior to each call to Cue() (but **after** any call to Reset()). It returns 0 if successful or, on failure, returns a negative error code listed in Appendix B.

long Save() – In the appropriate Registry tables, saves all of the current encoding properties for the currently selected encoder board. It returns 0 if successful, or, on failure, returns a negative error code listed in Appendix B. It is useful to call the Save() method after cueing for each encode, as the Load() or Cue() method may have made subtle changes to some of the Registry settings to guarantee a successful encoding session.

Registry Table Property Settings

Descriptions of the settings of Registry keys related to multi-board encoding are listed in the following tables. Each of the tables listed below corresponds to a property page in the MBProps application (see "MBProps," page 39).

The IBM Video Registry Table

The “IBM VideoX” table (where X represents the encoder board number) stores the encode properties that are related to the video elementary stream. Many of the video encoder properties are interrelated. The first of the following tables identifies and describes the properties themselves. The second table shows the relationship between format, chroma, mode, and resolution. Finally, the third table defines the relationship between I-frame distance (N value), reference frame distance (M value), closed/open GOP, and GOP structure.

In the table below, the Data Type identifies the recommended data type of the variable or class member that will be holding the property setting. Within the IBM Video Registry table itself, all keys are of type REG_DWORD.

IBM Video Registry Table				
Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**
Video Bit Rate	BitRate	Unsigned long	512,000 to 15,000,000 bps for 4:2:0 Chroma. Up to 50,000,000 bps for 4:2:2. Default: 8,000,000 bps.	Typically, SIF resolution (352x240/288) is used for lower bit rates (.5 to 3 Mbps). Half-D1 (352x480/576) is used for 3.5 to 6 Mbps. Full resolution (704 or 720 horizontal) is used for 7 Mbps and higher. If VBR mode is turned ON, BitRate represents the maximum video bit rate.

Variable Bit Rate Flag	VBRFlag	BOOL	Off = 0
			On = 1
			Default: 0
				Turns variable bit-rate mode on or off

Average Bit Rate	VBRAvgBitRate	unsigned long	Must be less than the BitRate setting if VBR is turned on.	If VBR is turned on, this setting defines the average video bit rate.

Table A-1. IBM Video Registry Table

IBM Video Registry Table (Continued)				
Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**

Video Mode	VideoMode	Unsigned char	VM_SIF = 0
			VM_AFF = 1
			Default: AFF
				VM_SIF represents MPEG-1
				VM_AFF represents MPEG-2.

Video Format	VideoFormat	Unsigned char	VF_NTSC=0
			VF_PAL=1
			Default: NTSC
				A setting of VF_NTSC is interpreted as NTSC; all other settings are interpreted as PAL.

Horizontal Resolution	HorizRes	Unsigned long	352 544 704 720 (Default)	See Table C-2 for acceptable combinations.
Vertical Resolution	VerticalRes	Unsigned long	120 (QSIF) 240 (SIF) 480 (Full) 512 (VBI) 288 (PAL SIF) 576 (PAL Full) 608 (PAL VBI) Default: 480	See Table C-2 for acceptable combinations.
Source Type	InputType	Unsigned char	0 – 8 Default: 0	A setting of 1 is interpreted as Digital, all others are interpreted as Composite.
Distance between I-frames (MPEG N value)	IFrameDistance	Unsigned long	1 – 16 Default: 15	See Table C-3.

Table A-1. IBM Video Registry Table (Continued)

IBM Video Registry Table (Continued)				
Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**

Distance between reference frames (MPEG M Value)	RefFrameDis-tance	Unsigned char	FS_IP = 1
			FS_IBP = 2
			FS_IBBP = 3
			Default: 3
				Where I and P are considered reference frames, the reference frame distance is defined as the number of frames from one reference frame up to but not including the next. It can also be seen as one more than the number of “B” frames between reference frames.

Chroma Format	ChromaFormat	Unsigned char	CF_4_2_0 = 0
			CF_4_2_2 = 1
			CF_4_4_4 = 2
			Default: 0
				A value of CF_4_2_2 is interpreted as 4:2:2. All other values are interpreted as 4:2:0.
				Note that Argus 4:2:0 encoders support only CF_4_2_0.

Closed GOP Flag	ClosedGOP	Unsigned long	0 = Open 1 = Closed Default: 0	See table C-3. Closed GOP setting is useful for post-encode editing.
Non-linear Quan-tization Flag	NonLinear-Quant	Unsigned long	0 = Off 1 = On Default: 0	Turns on/off the non-linear quantizer table. Use “1,” especially for low bit rates.
Concealment Vector Flag	Concealment-Vector	Unsigned long	0 = Off 1 = On Default: 0	Turn On to embed concealment vectors in the stream. Useful in noisy trans-mission environ-ments. Do not use with SIF.

Table A-1. IBM Video Registry Table (Continued)

IBM Video Registry Table (Continued)				
Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**
DC Precision	DCPrecision	Unsigned char	SIF: 8 MPEG-2: 9, 10, or 11. Default: 10	Number of bits used to represent the DC coefficients for intra-coded portions of pictures. See note (1), below.
Alternate Co-efficient Table	IntraTable	Unsigned char	0 = Off 1 = On Default: 0	A setting of 1 enables the alternate coefficient table, appropriate for MPEG-2 encodes. See note (2)
Aspect Ratio	AspectRatio	Unsigned char	1 = Square 2 = 4x3 3 = 16x9 4 = 2.21x1	Indicates aspect ratio of material being encoded.
NOTES: (1) The MPEG Specifications allow integral settings of 8 through 10. The IBM chip set also allows a non-standard setting of 11. The Filter Manager forces a setting of 8 whenever SIF resolution is specified, regardless of the value stored in the Registry. (2) In most cases, the IBM encoder will override this setting based on the compression type.				

Table A-1. IBM Video Registry Table (Continued)

Allowable Combinations of Video Properties

The table that follows lists acceptable combinations of video format, chroma, mode, horizontal resolution, and vertical resolution.

Format	**Chroma**	**Mode**	**Horizontal Res**	**Vertical Res**
NTSC	4:2:2	1 (MPEG-2)	720	512 (VBI)
	4:2:0	1 (MPEG-2)	720	512 (VBI)
	4:2:0	1 (MPEG-2)	720	480
	4:2:0	1 (MPEG-2)	704	480

Table A-2. Allowable Combinations of Video Properties

Format	Chroma	Mode	Horizontal Res	Vertical Res
	4:2:0	1 (MPEG-2) (Half-D1)	352	480
	4:2:0	0 (MPEG-1) (SIF)	352	240
PAL	4:2:2	1 (MPEG-2)	720	608 (VBI)
	4:2:0	1 (MPEG-2)	720	608 (VBI)
	4:2:0	1 (MPEG-2)	720	576
	4:2:0	1 (MPEG-2)	704	576
	4:2:0	1 (MPEG-2) (Half-D1)	352	576
	4:2:0	0 (MPEG-1) (SIF)	352	288

Table A-2. Allowable Combinations of Video Properties (Continued)

GOP Structure and Size

A GOP (group of pictures) is composed of a combination of I frames, B frames, and P frames. The only required frame type in a GOP is the I frame. If P and B frames are included in a GOP, they are arranged in repeated fixed sequences.

The multi-board Argus encoder allows from one to 16 frames per GOP. A GOP can be closed (it can be decoded by itself, with no reference to a previous or subsequent GOP) or open (it cannot stand alone). If the GOP is an open GOP, it is composed of an introductory I frame, followed by one or more of the following:

- From 0 to 15 “P” frames
- From 0 to 7 “BP” groups, followed by a single B at the end.
- From 0 to 14 “P” frames.

If the GOP is a closed GOP, it is composed of an introductory IP frame combination, followed by one of these:

- From 0 to 14 “P” frames
- From 0 to 6 “BP” groups, followed by a single “B” frame at the end
- From 0 to 4 “BBP” groups, followed by a “BB” pair at the end.

For all acceptable GOP structures, the I-frame distance (or N value in MPEG terminology) is defined as the number of frames between I frames, including the first, but excluding the second. The reference frame distance (or M value in MPEG terms) is defined as the number of frames between reference frames (where I and P are reference frames), including the first, but not including the second. Note that the introductory closed-GOP P frame is NOT considered when calculating the reference frame distance.

Let's look at some examples:

I-Frame Distance (N Value)	Ref-Frame Distance (M Value)	Open or Closed	GOP Structure
1	1	Either	II…(I-Only)
2	1	Either	IPIP…
4	1	Either	IPPPIPPP…
2	2	Open	IBIBIB…
6	2	Open	IBPBPBIBPBPB…
3	2	Closed	IPBIPBIPB…
7	2	Closed	IPBPBPBIPBPBPB…
3	3	Open	IBBIBBIBB…
6	3	Open	IBBPBBIBBPBB…
4	3	Closed	IPBBIPBB…
7	3	Closed	IPBBPBBIPBBPBB…

Table A-3. GOP Structure Examples

The *IBM Audio* Registry Table

The “IBM AudioX” table (where X represents the encoder board number) stores the encode properties that are related to the audio elementary stream. There can be up to two audio streams. In the following table, where the property is followed by the digit 0 or 1, the 0 or 1 specifies to which of the two audio streams the property applies.

In the table below, the Data Type identifies the recommended data type of the variable or class member that will be holding the property setting. Within the IBM Audio Registry table itself, all keys are of type REG_DWORD.

IBM Audio Registry Table				
Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**
Audio Bit Rate	BitRate0 BitRate1	Unsigned long	32,000 (mono) 48,000 (mono) 56,000 (mono) 64,000 80,000 96,000 112,000 128,000 160,000 192,000 224,000 256,000 320,000 384,000 Default: 192,000	
Audio Sample Rate	SampleRate0 SampleRate1	Unsigned long	32,000 44,100 48,000 Default: 48,000	

Table A-4. IBM Audio Registry Table

IBM Audio Registry Table (Continued)				
Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**
Audio Mode	Mode0 Mode1	Unsigned char	Stereo = 0, Joint Stereo = 1 Dual Audio = 2, Single Audio = 3, Multiple = 4 Default: Stereo	

Audio Input	Input0
	Input1
		Unsigned char	Analog = 0
			Digital = 1
			Inactive = 2
			Default: Analog
				Second audio stream is set to inactive if it is not being used.

Error Protect Flag	ErrorProtectFlag0 ErrorProtectFlag1	BOOL	0 = FALSE 1 = TRUE Default: FALSE	Refers to a setting in an MPEG audio header. NOTE: Use with extreme caution. A setting of 1 may corrupt the PTS.
Copyright Flag	CopyrightFlag0 CopyrightFlag1	BOOL	0 = FALSE 1 = TRUE Default: FALSE	Refers to a setting in an MPEG audio header.
Original Flag	OriginalFlag0 OriginalFlag1	BOOL	0 = FALSE 1 = TRUE Default: FALSE	Refers to a setting in an MPEG audio header. Marks stream as original or copy.
Audio Slave Mode	WaitOnStartFlag0 WaitOnStartFlag1	BOOL	0 = FALSE 1 = TRUE Default: TRUE	To guarantee A/V synchronization, should be set to 1 when both video and audio are being encoded. Then the audio start is triggered by the start of the video encoder.
Audio Head Room	HeadRoom0 HeadRoom1	Unsigned long	18 (Default) 20	

Table A-4. IBM Audio Registry Table (Continued)

IBM Audio Registry Table (Continued)				
Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**
Audio Reference Level	ReferenceLevel0 ReferenceLevel1	Unsigned char	0 = +4dB 1 = 0dB 2 = -10dB	+4 dB is default value. Configurable reference level is available only with encoder firmware version 3.0 or later.
NOTE: (1) In previous releases, the last five entries in the IBM Audio Registry Table were stored in the ArgusConfig.txt configuration file. The configuration file is no longer used in versions 2.3 and later. All of its entries have been moved to the Windows Registry.				

Table A-4. IBM Audio Registry Table (Continued)

The *Mux* Registry Table

The "MuxX" table (where X represents the encoder board number) stores the encode properties that are related to the multiplexing of the audio and video elementary streams into a single system, program, or transport stream. Note, however, that it is possible to generate "unmixed" video and audio elementary streams by setting the MPEG Standard to 3.

In the table below, the Data Type identifies the recommended data type of the variable or class member that will be holding the property setting. Within the Mux Registry table itself, all keys are of type REG_DWORD.

Mux Registry Table				
Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**
Stream Type	MPEGStd	Unsigned char	0 = System 1 = Program 2 = Transport 3 = Elementary Default: 2	Type of multiplexed stream being generated by this encode.
Mux Rate	MuxRate	Unsigned long	Range is 1,500,000 to 50,000,000 Default: 8,000,000	If the MPEGStd is set to 2, this key represents the overall bit rate of the transport stream. Otherwise, it has no meaning. (For system, program, and elementary streams, the audio and video bit rates determine the overall mux rate).
Closed Caption Flag	ClosedCaption-Flag	BOOL	TRUE if mux must insert closed captioning, FALSE otherwise. Default: FALSE	If the resolution is set to 720 VBI, this flag is set to 0, because the closed caption is encoded with the VBI.

Table A-5. Mux Registry Table

Mux Registry Table (Continued)				
Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**
Closed Caption Format	ClosedCaption-Format	Unsigned long	0 = C-Cube 1 = ATSC 2 = Reordered C-Cube 3 = ATSC reordered	See note (1).
Adjust GOP Time Code Flag	AdjustGopTime-Code	Unsigned char	0 = Off 1 = On Default: 0	A setting of 1 tells the encoder to turn on GOP time code adjustment. See note (2).
Starting GOP Time Code	GopTcStart	Unsigned long	Default: 0	If the adjust GOP time code flag is set to 1, this key identifies the starting time code. See note (3).
Duration of the Encode	Duration	Unsigned long	Default: 900	The proposed duration of the encode, in frames.
Audio Stream ID	AudioStreamID0 AudioStreamID1	Unsigned long	Allowable range: 0–31. Default is 0 for first audio stream, 1 for second audio stream.	PES header stream ID of audio stream.
Video Stream ID	VideoStreamID	Unsigned long	Allowable range: 0–15. Default: 0.	PES header stream ID of the video stream.
Audio Stream PID	AudioStreamPID0 AudioStreamPID1	Unsigned long	Minimum value: 16. Minimum DVB-compliant value: 512. Default: 640,641	Must be unique among all other audio and video PIDs for this stream.
Video Stream PID	VideoStreamPID	Unsigned long	Minimum value: 16. Minimum DVB-compliant value: 512. Default: 512	Must be unique among all other audio and video PIDs for this stream.

Table A-5. Mux Registry Table (Continued)

Mux Registry Table (Continued)				
Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**
Audio Packet Size	AudioPktSize	Unsigned long	Default: 1728	Size (in bytes) of an audio PES packet. May be overridden, especially for transport streams.
Video Packet Size	VideoPktSize	Unsigned long	Default: 1728	Size (in bytes) of a video PES packet. May be overridden, especially for transport streams.
Language Code	Audio Language Code0 Audio Language Code1	Short	0 = English 1 = Spanish 2 = French 3 = German 4 = Japanese 5 = Dutch 6 = Danish 7 = Finnish 8 = Italian 9 = Greek 10 = Portuguese 11 = Swedish 12 = Russian 13 = Chinese	This code is informational only, used to identify the language in which the audio is presented.
First System Clock Reference	FirstClockRef	Unsigned long	Default: 0	First system-clock value to be assigned by mux component.

Table A-5. Mux Registry Table (Continued)

Mux Registry Table (Continued)				
Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**
NOTES: (1) The “reordered” notation instructs the encoder to sort the bytes of closed caption data so that they are actually stored in the frames on which they will be displayed. Otherwise, the decoder will sort the closed caption data to put it in the correct display-order. “0” is the setting for the standard C-Cube closed caption format. “3” is the setting for the standard ATSC closed caption format. Neither of the two ATSC formats is supported for SIF encodes. Note that encoder firmware version 1.20 or later is required to use settings “2” or” 3.” (2) Turning on GOP-time-code adjustment instructs the encoder to stamp the GOP time codes in such a way that the time code of the first GOP is equal to the setting of the GopTcStart key, defined in the table above and in note (3). The time codes of all subsequent GOPs are then offset by the GopTc-Start setting. (3) The starting time code is an unsigned long of the format t:hh:mm:ss:ff, where the high-order digit “t” represents the time code type (0 = PAL, 1 = NTSC, 2 = drop-frame NTSC); the “hh” digits represent the hours field of the time code, the “mm” digits represent the minutes field, the “ss” digits represent the seconds field, and the “ff” digits represent the frames field. For example, a drop-frame starting time code of 01:32:43:14 would be represented as 201324314.				

Table A-5. Mux Registry Table (Continued)

The *MuxStore* Registry Table

The "MuxStoreX" table (where X represents the encoder board number) stores the encode properties that determine whether, how, and where the output muxed file is stored.

In the table below, the Data Type identifies the recommended data type of the variable or class member that will be holding the property setting. Within the MuxStore Registry table itself, the two keys marked BOOL are of type REG_DWORD; the FileName key is of Registry type REG_MULTI_SIZE.

MuxStore Registry Table

Property	Registry Key	Data Type	Value Set	Comments
File Storage Enabled	StorageEnabled	BOOL	0= Disabled 1= Enabled Default: 1	See note (1).
File Name	LocalFilename	CString		Must be a legitimate path name recognized by the encoder.
Optimize the Writes?	OptimizedMux-Writes	BOOL	0 = Do not optimize 1 = Optimize Default: 1	See note (2).

NOTES: (1) When the MPEG stream type is system, program or transport, you will probably be creating and storing a muxed file, so this setting should be "1." However, if you are generating elementary streams, this setting should be "0." Instead, set the enabled flag in the VideoStore and AudioStore tables to "1."

(2) Turning optimization on makes the file-writes more efficient. However, a side effect of turning optimization on is that the file size will not be recognized by the operating system until the end of the encode, when the application closes the file. If you need to see a file size that increases as the encode progresses, turn optimization off, but be aware that the encode may fail if the file I/O cannot keep up with the speed of the encoder.

Table A-6. Mux Store Registry Table

The *VideoStore* Registry Table

The "VideoStoreX" table (where X represents the encoder board number) stores the encode properties that determine whether, how, and where the output video elementary file is stored (elementary-stream encode only).

In the table below, the Data Type identifies the recommended data type of the variable or class member that will be holding the property setting. Within the VideoStore Registry table itself, the two keys marked BOOL are of type REG_DWORD; the FileName key is of Registry type REG_MULTI_SIZE.

VideoStore Registry Table

Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**
File Storage Enabled	StorageEnabled	BOOL	0= Disabled 1= Enabled Default: 1	See note (1).
File Name	LocalFilename	CString		Must be a legitimate path name recognized by the encoder.
Optimize the Writes?	OptimizedMux-Writes	BOOL	0 = Do not optimize 1 = Optimize Default: 1	See note (2).

NOTES: (1) When the MPEG stream type is elementary, you will probably be creating and storing a video and audio file, so this setting should be "1." However, if you are generating a system, program, or transport stream, this setting should be "0." Instead, set the enabled flag in the MuxStore table to "1."

(2) Turning optimization on makes the file-writes more efficient. However, a side effect of turning optimization on is that the file size will not be recognized by the operating system until the end of the encode, when the application closes the file. If you need to see a file size that increases as the encode progresses, turn optimization off, but be aware that the encode may fail if the file I/O cannot keep up with the speed of the encoder.

Table A-7. Video Store Registry Table

The *FirstAudStore* Registry Table

The "FirstAudStoreX" table (where X represents the encoder board number) stores the encode properties that determine whether, how, and where the output file for the first audio stream is stored (elementary-stream encode only).

In the table below, the Data Type identifies the recommended data type of the variable or class member that will be holding the property setting. Within the FirstAudStore Registry table itself, the two keys marked BOOL are of type REG_DWORD; the FileName key is of Registry type REG_MULTI_SIZE.

FirstAudStore Registry Table				
Property	**Registry Key**	**Data Type**	**Value Set**	**Comments**
File Storage Enabled	StorageEnabled	BOOL	0= Disabled 1= Enabled Default: 1	See note (1).
File Name	LocalFilename	CString		Must be a legitimate path name recognized by the encoder.
Optimize the Writes?	OptimizedMux-Writes	BOOL	0 = Do not optimize 1 = Optimize Default: 1	See note (2).
NOTES: (1) When the MPEG stream type is elementary, you will probably be creating and storing a video and audio file, so this setting should be "1." However, if you are generating a system, program, or transport stream, this setting should be "0." Instead, set the enabled flag in the MuxStore table to "1." (2) Turning optimization on makes the file-writes more efficient. However, a side effect of turning optimization on is that the file size will not be recognized by the operating system until the end of the encode, when the application closes the file. If you need to see a file size that increases as the encode progresses, turn optimization off, but be aware that the encode may fail if the file I/O cannot keep up with the speed of the encoder.				

Table A-8. First Audio Store Registry Table

The *SecondAudStore* Registry Table

The “SecondAudStoreX” table (where X represents the encoder board number) stores the encode properties that determine whether, how, and where the output file for the second audio stream, if there is one, is stored (elementary-stream encode only).

In the table below, the Data Type identifies the recommended data type of the variable or class member that will be holding the property setting. Within the SecondAudStore Registry table itself, the two keys marked BOOL are of type REG_DWORD; the FileName key is of Registry type REG_MULTI_SIZE.

SecondAudStore Registry Table

Property	Registry Key	Data Type	Value Set	Comments
File Storage Enabled	StorageEnabled	BOOL	0= Disabled 1= Enabled Default: 1	See note (1).
File Name	LocalFilename	CString		Must be a legitimate path name recognized by the encoder.
Optimize the Writes?	OptimizedMux-Writes	BOOL	0 = Do not optimize 1 = Optimize Default: 1	See note (2).

NOTES: (1) When the MPEG stream type is elementary, you may or may not be creating and storing a second audio stream, as well as the primary audio and video streams. If you are encoding two audio streams and wish to store both, set this key to “1.” However, if you are generating a system, program, or transport stream, or if you are generating just a single audio elementary stream, this setting should be “0.”

(2) Turning optimization on makes the file-writes more efficient. However, a side effect of turning optimization on is that the file size will not be recognized by the operating system until the end of the encode, when the application closes the file. If you need to see a file size that increases as the encode progresses, turn optimization off, but be aware that the encode may fail if the file I/O cannot keep up with the speed of the encoder.

Table A-9. Second Audio Store Registry Table

Appendix B

Filter Manager Error/Status Codes

The following return codes may be returned by calls to Filter Manager methods or by Filter Manager Error Events. This table applies to both the Argus single-board encoder system and the Argus multi-board encoder.

NOTE 1: There are six error codes that are reported by the video encoder chip and passed unaltered through Filter Manager. Unfortunately, these six error codes conflict with identically numbered errors generated by Filter Manager itself. Although it is true that these six error codes do not uniquely identify a single specific Argus error condition, in most cases the error condition can be identified by the context in which it is reported. The error codes reported by the encoder chip are identified as such in the Comments section of the Error/Status Codes table that follows.

NOTE 2: If you abort an encode or shut down your application without cleanly ending an encode, you must make certain that CVProServer and MemMgrServer have both been terminated before you restart the application. You can terminate these services using Task Manager.

Filter Manager Error/Status Codes		
Error Code	**Meaning**	**Comments**
11	Encode has been successfully resumed after a pause.	Not an error.
0	Operation completed successfully. Status OK.	Not an error.
-5	V_NO_DATA	**(Encoder chip error code)** No video data was available at the time of the last "read." Run encoder diags, check the video source.
-9	Unable to eject tape using Sony command.	
-10	Video component failed to start, stop, pause, resume, or reset.	Summary error message. Check log for more descriptive messages regarding status of video encoder.
-12	Mux component failed to start, stop, pause, resume, or reset.	Summary error message. Check log for more descriptive messages regarding status of mux.

Table B-1. Filter Manager Error/Status Codes

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-13	Main storage component failed to stop, pause, or resume.	Summary error message. Check log for more descriptive messages regarding status of storage component.
-13	V_FIFO_UNDERFLOW	**(Encoder chip error code)** A timeout occurred while attempting to read data from the encoder. Run encoder diags, check the video source.
-14	CineView Pro component failed on a stop.	May be locked up. Check log for other error messages.
-14	V_FIFO_OVERFLOW	**(Encoder chip error code)** A FIFO overflow was detected when trying to read data from the board. May be the result of setting the bitrate too high, given the configuration of the system and the nature of the encode. Also, make certain no other process is running on the system at the time of the encode.
-15	VTR component failed to cue, stop, pause, or write adjustment to Registry.	Summary message. Check log for other VTR-related messages.
-15	V_FIFO_READ	**(Encoder chip error code)** An error occurred while attempting to read the video FIFO. Run encoder diags, check the video source.
-16	Storage component for elementary audio or video stream failed to stop or resume.	Summary message. Should occur only during elementary stream encode.

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-16	V_BITRATE	**(Encoder chip error code)** Too little video data available at the time of the last "read." Run encoder diags, check the video source.
-17	Disk space error.	Application determined that there is insufficient disk space for the muxed file or for the elementary video file.
-18	VSP component failed during a reset.	Summary message. Check for more specific information from VSP component.
-25	Exception thrown during reset, cue, or start.	
-26	Failure to create one of Filter Manager mutexes.	Probably a system error. Check number of open handles.
-27	Filter manager failed when trying to initialize COM libraries.	System problem? COM DLLs not installed?
-28	Multi-stream encode error message.	User should not attempt multi-stream encodes with Argus 4:2:2/4:2:0 software.
-29	CineViewPro component failed to reset, cue, or start.	Summary message. Check for more specific messages from decoder component.
-30	No mux filename was supplied, so Filter Manager failed to create a codec file name.	Check file name field to make sure that it contains a legitimate file name.
-31	Exception thrown loading parameters from the Registry.	Summary message. Check Argus Registry to see if table is there.
-32	No longer used.	

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-33, -34	Error resetting or starting audio storage component.	Summary message. Check log file for more specific messages. Last encode may not have shut down correctly? Exit application and try again.
-35	Multi-stream encode error message.	User should not attempt multi-stream encodes with Argus 4:2:2/4:2:0 software.
-36	Main storage component failed to start or reset.	Summary message. Check for more specific messages in error log.
-39	Unable to set outpoint on tape deck.	Check COM port connection (including converter). Make sure tape deck is on remote. Make sure application is looking at the correct COM port.
-41	Invalid VTR shuttle speed requested through Sony protocol.	
-42	Tape deck failed to receive shuttle command.	Check COM port connection (including converter). Make sure tape deck is on remote. Make sure application is looking at the correct COM port.
-44	Invalid pre-roll value. Pre-roll should be a number between 0 and 60 (seconds). AND mark-in value must be greater than pre-roll.	Check validity of mark-in and preroll.
-45	When you adjust the mark-in time code by adding or subtracting the pre-roll adjustment, the result is invalid (<0).	Check mark-in and adjustment time codes. Adjustment must be less than mark-in.
-47	Unable to read preroll from tape deck.	Check COM port connections, Check to see that VTR is in remote mode, etc.

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-48	Unable to read mark-in from tape deck.	Check COM port connections, Check to see that VTR is in remote mode, etc.
-49	Unable to recognize microcode type in looking up VTR adjustment in Registry.	
-50	Error initializing pin that connects IBMVideo to Mux component.	Check swap space on C:\ drive. Make sure mux was shut down properly last time (that CVPro Server and MemMgrServer were not left running).
-51	Attempted to initialize a video component that was already initialized.	Try resetting or quitting application. Make sure no other instances of the encoder are running simultaneously. Be sure to terminate CVProServer and MemMgrServer before restarting.
-52	Attempted to cue a video component that was already cued.	Try resetting or quitting application. Make sure no other instances of the encoder are running simultaneously. Be sure to terminate CVProServer and MemMgrServer before restarting.
-52	V_INIT	**(Encoder chip error code)** An error occurred while attempting to initialize the video board. Run diags.
-53	Attempted to cue a video component that was already playing.	Terminate last encode (or wait for it to finish) before starting next encode.

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-54	Attempted to Pause the video component while it was already paused.	Application may have lost track of its state. If error persists, you may need to exit application and restart.
-55	Attempted to Start the video component before cueing it.	Application appears to have lost track of its state. If error persists, you may need to exit application and restart.
-56	Attempted to Stop the video component, although it was not currently playing.	Application appears to have lost track of its state. If error persists, you may need to exit application and restart.
-57	Attempted to Resume the video component, although it was not currently paused.	Application appears to have lost track of its state. If error persists, you may need to exit application and restart.
-60	An exception was thrown from within the video process thread.	Check video settings (PAL/ NTSC, Digital/ Composite). Check video encoder hardware. Run video encoder diagnostics.
-61	Driver command to start video returned unsuccessfully OR exception was thrown by video Start method.	Check video settings (PAL/ NTSC, Digital/ Composite). Check video encoder hardware. Run video encoder diagnostics.
-70	Audio chip timed out during a read or shut-down command.	This is either a problem with the audio encoder hardware or with the system.
-71	Attempting to reinitialize an already-initialized audio component.	Shut down application. Make sure CVProServer and MemMgrServer have been terminated. Restart.

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-73	Attempted to cue or Start an audio component that is currently encoding.	The application may have hung up during the previous encode. Try calling Reset. If that fails, restart the application.
-74	Attempted to Pause the audio component when it was already paused.	State problem? May need to restart the application.
-75	Attempted to Start the audio component before cueing it.	State problem? May need to restart the application.
-76	Attempted to Stop or Pause the audio component although it is not playing.	State problem? May need to restart the application.
-77	Attempted to Resume the audio component, although it is not paused.	State problem? May need to restart the application.
-78	Encode failed because audio pin over- flowed, or application was not successful in creating, initializing, or resetting the pin from the audio component to the mux.	If error occurred when application was coming up, check the C drive to be sure it has adequate swap space. If error occurred during an encode, the audio pin backed up-this is usually a secondary error. The audio backup is caused because some other component failed or “hung up” the encoder. Shut down application, terminating CVProServer and MemMgrServer if needed.
-79	A stop was issued to the audio component, but it won't stop in a reasonable amount of time.	It may be hung up in a while-loop. May need to terminate application with task manager.
-80	Software failed trying to read the firmware revision OR exception was thrown during the audio component initialization process.	Check audio encoder hardware. Make sure board is installed properly.

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-81	Audio process failed because of a driver initialization error, an error reading data from the audio board, or for some other non-pin-related error.	Check audio encoder hardware. Make sure board is installed properly.
-82	The start-audio driver command failed or an exception was thrown during the audio Start method.	Check audio encoder hardware. Make sure board is installed properly.
-83	The init_audio driver command failed OR an exception was thrown during the audio Cue method.	Check audio encoder hardware. Make sure board and software are installed properly.
-98	While checking Registry for adjustment value, VTR encountered an invalid microcode type designator.	Check to be sure that full set of current software was installed successfully.
-113	Exception thrown by FTP component while streaming data.	
-114	No FTP server name was provided, or an exception was thrown trying to connect to FTP server.	Check Registry to make certain that FTP server name was provided if you asked for a streaming encode. Then check FTP connections and server setup.
-115	Error establishing internet session for FTP transfer.	
-116	Either remote file name was not filled in or there was an exception thrown while trying to open the remote (FTP) file.	
-117	Error opening local storage file.	Check path name, folder permissions. Make sure file not already open.
-118	Error writing to or closing local file.	Check disk fullness and disk status. If this is a high-bitrate encode, it may be that the disk can't handle the throughput.
-119	Error initializing the input pin of the storage component.	

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-121	Error creating the storage component process thread.	
-122	Encoder is unable to communicate through the serial port with the VTR.	Check the cabling from the VTR to the converter, from the converter to the serial port (make sure converter is not in backwards). Check the COM port setting on the encoder application (COM1 or COM2) and make sure it matches the number of the serial port being used. Make sure that the VTR is turned on and that it is in remote mode.
-165	Error communicating with the VSP.	
-166	Error creating CVspApi class.	Mismatched software components?
-170	Attempt to call audio Get or Put method with a stream index other than 0 or 1, or attempting to set invalid audio bitrate, invalid audio input type, audio layer, or audio headroom with a Put() call.	Programming error. Check source code.
-175	Error creating playback COM object.	Ascertain that CinProSerCom is registered.
-177	Error creating VTR COM object.	Ascertain that VtrControl is registered.
-179	Error creating Audio COM object.	Ascertain that IBMAudio is registered.
-181	Error creating Video COM object.	Ascertain that IBMVideo is registered.
-183	Error creating audio elementary storage object.	Ascertain that RemoteStore is registered.
-185	Error creating video elementary storage object.	Ascertain that RemoteStore is registered.

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-189	Error creating mux storage object.	Ascertain that RemoteStore is registered.
-194	Multi-stream encode error message.	User should not attempt multi-stream encodes with Argus 4:2:2/4:2:0 software.
-201	Error creating VSP object.	Ascertain that IBMVSP is registered.
-219	Error creating second audio object.	Ascertain that IBMAudio is registered.
-227	Multi-stream encode error message.	User should not attempt multi-stream encodes with Argus 4:2:2/4:2:0 software.
-230	During cue, mux component received an invalid stream type (0=system, 1=program, 2=transport, 3=elementary).	Check Registry setting of mux stream type.
-233	Error creating or initializing plug-in component's input pin.	-233 through -248 are all plug-in errors. Since they are working with the source code, developers should be able to track these error codes themselves.
-234	Attempted to initialize plug-in component when it was already initialized.	
-235	Error starting suspended plug-in process thread.	
-236	Attempted to start plug-in when it is already encoding.	
-237	Attempted to start plug-in without cueing it.	
-238	Attempted to stop the plug-in component when it was not playing.	
-239	Attempted to pause the plug-in component when it was already paused.	

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-240	Attempted to resume the plug-in component when it was not paused.	
-241	Plug-in component failed while writing to file.	
-242	Plug-in failed to open file or allocate resources.	
-243	Attempted to cue plug-in when it was already cued.	
-245	Error creating one of the plug-in objects.	
-247	The Initialize() method failed for one of the plug-ins.	
-248	Summary error code. Filter manager failed in Cueing, Starting, Stopping, Pausing, Resuming, or Resetting a plug-in.	
-250	A pin underflowed. The error message will indicate which pin.	This message indicates that one of the components is starved for data (it's not being delivered fast enough).
-251	A pin overflowed. The error message will indicate which pin.	Usually this is an indication that the system is not able to handle the volume of work that it is being asked to accomplish. Check task manager during encode to see where the bottleneck might be.
-252	Unable to find a matching reader/writer for a specified pin.	
-253	An attempt was made to read or to write too large a block of data to/from a pin. (Block was larger than pin size).	The error message will indicate which pin. This is a programming error.
-254	An attempt to create a pin object failed. The error message will indicate which pin.	Ascertain that Vela_Pin is registered.
-255	Pin component failed when trying to create a mutex	System error. Check handle count using task manager.

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-260	Undefined error occurred when attempting to create or use a pin.	
-331	RemoteStore component attempted to write very last block of data to the file, but failed. This error occurs only when the FilterMgr “Optimized MuxWrite” flag is set to 1.	We use a write procedure that requires that all write-blocks must be evenly divisible by the disk sector size. To get around this restriction on the last block of data, we close the file, reopen it in another mode, then write the last block. This error could be a timing error-make certain that no attempt was made to move or lock the file before the encode finished.
-332	Error closing the remote (FTP) file.	Make certain that the FTP process was not aborted before the encode finished.
-334	Failure creating, initializing, or using the decoder input pin (usually from the Mux).	
-335	The PlayFromPin call to the CVPro Server failed. The decoder failed to start realtime playback.	Make sure that the previous encode did not end with an unclosed CVProServer or MemMgrServer executable running. Make sure that the decoder board is installed properly and functioning properly. Check firmware / hardware revisions of the decoder board.
-337	Attempt to set up CVPro scaler failed.	See No. -335.
-338	Attempted to stop the decoder when it was not playing.	
-340	An invalid closed caption-type was defined (read from Registry).	See notes in Appendix A on closed caption types.

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-341	Unable to create or initialize CVProServer object.	Ascertain that CVProServer is registered and functional.
-342	CVPro failed on request to initialize.	Check functionality of decoder in general using standard CVPro client application.
-343	CVProServer failed to pause.	
-344	CVProServer failed to resume.	
-345	When the mux Start component was called, there was no active thread running to start.	State problem. May need to restart application, making certain that CVProServer and MemMgrServer are terminated.
-346	FilterManager asked mux component to create an undefined stream type. (See -230).	
-347	Mux component failed to open the mux writer stream.	
-348	An GOP Size of 0 or less was passed to Mux.	Check the I-frame distance setting in the Video Registry.
-349	Mux component failed to create its mutexes.	System problem. Check number of open handles using task manager.
-350	Mux failed when trying to initialize the closed caption “driver” class.	
-400 … -438	MPEG-1 multi-stream error.	User should not attempt multi-stream encodes with Argus 4:2:2/4:2:0 software.
-440	Audio storage component (during elementary encode) failed to pause.	
-441	During elementary encode, video component failed to pause.	

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-443	If a system, program, or transport stream is selected as the mux type in the mux Registry, but the mux-file-enabled flag is not set in the "Filter-Mgr" Registry table, this error flag is set. Also, if an elementary stream is selected as the mux type in the mux Registry, but the video-file-enabled flag is not set in the "FilterMgr" Registry, this flag is set.	Check stream type in Mux Registry, and compare it to the file type enabled in the Filter-Mgr Registry. See Appendix A.
-444	Invalid mux file path name.	Make sure that the pathname specified for the mux file is present and writable.
-445	Invalid video file path name.	For an elementary-stream encode, make sure that the pathname specified for the video file is present and writable.
-446	Invalid audio file path name.	For an elementary stream encode, make sure that the pathname specified for the audio file is present and writable.
-448	Argus Registry failure.	Unable to open the HKEY_CURRENT_USER path Software\Vela Research\Broadcast Argus. Check the Registry.
-449	Unable to open the "IBM Video" Registry table.	
-450	Unable to open the "IBM Audio" Registry table.	
-451	Unable to open the "Mux" Registry table.	
-452	Unable to open the "DualEnc" Registry table	
-455	Unable to open the "VTR" Registry table.	
-456	Unable to open the "RemoteStore" Registry table.	
-457	Unable to open the "FilterMgr" Registry table.	

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-458	Invalid video bitrate supplied.	Check IBM Video Registry table (see Appendix A). Must be between 512,000 and 50,000,000. May not exceed 3,500,000 for SIF. May not exceed 15,000,000 for 4:2:0 chroma.
-459	Invalid horizontal resolution supplied for main encode. Must be 352, 480, 544, 704, or 720. (352 is the only valid value for SIF.)	Check IBM Video Registry table (see Appendix A).
-460	Invalid vertical resolution supplied for main encode. Must be 120, 240, 480, 512 for NTSC or 144, 288, 576, 608 for PAL. (For SIF, must be 240 or 288.)	Check IBM Video Registry table (see Appendix A).
-461	Invalid video mode supplied for main encode. Must be SIF (0) or AFF (1)	Check IBM Video Registry table (see Appendix A).
-462	Invalid video format. Must be NTSC (0) or PAL (1).	Check IBM Video Registry table (see Appendix A).
-463	Inverse telecine is not supported.	
-464	Invalid input type supplied. Must be 1 for digital or any other value between 0 and 8 for composite.	Check IBM Video Registry table (see Appendix A).
-465	Invalid I-frame distance supplied in IBM Video Registry. Note that the I-Frame distance must agree with the RefFrameDistance and the Closed GOP flag.	Check IBM Video Registry table (see Appendix A) as well as section that immediately follows, also in Appendix A, explaining relationship between Closed GOP, I-frame distance, and ref-frame-distance.
-466	Invalid RefFrameDistance in IBM Video Registry table.	Check IBM Video Registry table (see Appendix A).
-467	Invalid ClosedGOP setting in IBM Video Registry table.	Check IBM Video Registry table (see Appendix A).

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-468	Invalid chroma setting in IBM Video Registry table.	Check IBM Video Registry table (see Appendix A).
-469	Embedded metadata is not supported.	
-470	Invalid non-linear quantization setting. Must be 0 or 1.	Check IBM Video Registry table (see Appendix A).
-471	Invalid Concealment Vector setting. Must be 0 or 1. MUST set to 0 for low-bitrate SIF, or corrupted macroblocking will occur.	Check IBM Video Registry table (see Appendix A).
-472	Invalid DC Precision setting. Must be 8, 9, 10, or 11, with the value of 8 reserved exclusively for SIF encodes. FilterManager will automatically encode SIF with a DCPrecision of 8, regardless of setting in Registry.	Check IBM Video Registry table (see Appendix A).
-473	Invalid Intra-table flag.	Check IBM Video Registry table (see Appendix A).
-474	Invalid aspect ratio. Must be square(1), 4x3 (2), 16x9 (3) or 2.21 x 1 (4).	Check IBM Video Registry table (see Appendix A).
-475	Invalid audio bitrate supplied for main encode. Must be 32000, 48000, 56000, 64000, 80000, 96000, 112000, 128000, 160000, 192000, 224000, 256000, 320000, or 384000.	Check IBM Audio Registry table (see Appendix A).
-476	Invalid sample rate supplied for main encode. Must be 32000, 441000, or 48000.	Check IBM Audio Registry table (see Appendix A).
-477	Invalid audio mode supplied for main encode. Must be 0-Stereo, 1-Joint, 2-Dual, 3-Single.	Check IBM Audio Registry table (see Appendix A).
-478	Invalid audio input supplied for main encode. Must be analog, digital, or inactive.	Check IBM Audio Registry table (see Appendix A).
-479	Invalid audio layer for main encode. We support only layer 2.	Check IBM Audio Registry table (see Appendix A).
-480	Invalid error protect flag supplied for main encode. Must be 0 or 1.	Check IBM Audio Registry table (see Appendix A).

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-481	Invalid copyright flag supplied for main encode. Must be 0 or 1.	Check IBM Audio Registry table (see Appendix A).
-482	Invalid “original” flag supplied for main encode. Must be 0 or 1.	Check IBM Audio Registry table (see Appendix A).
-483	Invalid slave-mode setting for main encode. Must be set to 1 if both audio AND video streams are enabled.	Check IBM Audio Registry table (see Appendix A).
-484	Invalid audio headroom setting for main encode. Must be 18 or 20.	Check IBM Audio Registry table (see Appendix A).
-485	Invalid audio stream ID for main encode. Must be a value of 0 to 31. May not duplicate stream ID of other AUDIO streams in this encoded file.	Check Mux Registry table (see Appendix A).
-486	Invalid video stream ID for main encode. Must be a value between 0 and 15.	Check Mux Registry table (see Appendix A).
-487	Invalid mux stream type for main encode. Must be system (0), program (1), transport (2) or elementary (3).	Check Mux Registry table (see Appendix A).
-488	Invalid language setting for one of audio streams in main encode.	Check Mux Registry table (see Appendix A).
-489	Invalid audio PID for one of audio streams in main encode. Valid only for transport stream. Must be between 0x10 and 0x1fff. Must be unique among all component streams of transport stream.	Check Mux Registry table (see Appendix A).
-490	Invalid video PID for video stream in main encode. Valid only if this is a transport stream. Must be between 0x10 and 0x1fff. Must be unique among all component steams of transport stream.	Check Mux Registry table (see Appendix A).
-491	Invalid setting for “Adjust GOP Time code” flag. Must be off (0) or on (1). Can be turned on only if VTR-control is turned on.	Check Mux Registry table (see Appendix A).

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-492	Invalid mux rate supplied for main encode. Used only for transport stream. Must be between 512000 and 50,000,000.	Check Mux Registry table (see Appendix A).
-493	Invalid closed caption flag setting for main encode. Must be off (0) or on (1).	Check Mux Registry table (see Appendix A).
-494	Invalid closed caption format. Must be between 0 and 3. Meaningful only if closed caption flag is set to 1.	Check Mux Registry table (see Appendix A).
-495	Invalid SourceEnabled setting. Mus4t be 0 to turn OFF VTR Control, or 1 to turn it on. Can be turned on only if VTR is marked as installed in Encoder Config Registry.	Check VTR Registry table (see Appendix A).
-496	Invalid Com-port setting for VTR component. Must be 1 or 2 and must represent serial port through which encoder communicates with VTR.	Check VTR Registry table (see Appendix A).
-497	Invalid VTR adjustment for VTR component. Must be between -20 frames and + 20 frames.	Check VTR Registry table (see Appendix A).
-498	Invalid pre-roll for VTR. Must be >= 0 if VTR control is enabled. Represents number of frames earlier (-) or later (+) to start encode.	Check VTR Registry table (see Appendix A).
-499	Invalid drop frame setting. Must be 0 or 1 if this is NTSC content, or 0 if it is PAL. This value will be overridden during the encode with the actual drop-frame setting of the tape once the encode is cued and/or started.	Check VTR Registry table (see Appendix A).
-500	Invalid segment count. You must define at least one and no more than 3 durations. If VTR-control is enabled, the duration is represented by a mark-in and mark-out pair. If VTR-control is disabled, the duration is represented by the Duration time code. All of these are defined in the VTR Registry table.	Check VTR Registry table (see Appendix A).
-501	Invalid Mark-in time code (used only when VTR-Control is turned on).	Check VTR Registry table (see Appendix A).

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-502	Invalid Mark-out time code (used only when VTR-Control is turned on).	Check VTR Registry table (see Appendix A).
-503	Invalid duration (used only when VTR-Control is turned off).	Check VTR Registry table (see Appendix A).
-504	Invalid setting of mux-file-enabled flag (which determines if a muxed file is to be stored during a system, program, or transport stream encode), or of the video-file-enabled flag or audio-file-enabled flag (which determines if the video or audio file is to be stored for an elementary-stream encode). The mux file cannot be enabled for an elementary encode, nor can the video or audio file be enabled for a muxed encode.	Check FilterMgr Registry table (see Appendix A).
-505	Invalid mux file name for main MPEG_2 file. A file name must be supplied for the stored file if the mux-file-enabled flag is turned on.	Check FilterMgr Registry table (see Appendix A).
-506	Invalid video file name supplied for an elementary-stream encode where video-file-enabled flag is turned on.	Check FilterMgr Registry table (see Appendix A).
-507	Invalid audio file name supplied for an elementary-stream encode where audio-file-enabled flag is turned on.	Check FilterMgr Registry table (see Appendix A).
-508	No file store selected. Either the mux-file-enabled or the video-file-enabled flag must be selected.	Check FilterMgr Registry table (see Appendix A).
-509	Invalid playback-enabled flag. It must be set to 0 or 1.	Check FilterMgr Registry table (see Appendix A).
-510	One of the multi-encode selections was turned on when the encode type selected was elementary stream, or when playback was turned off.	Check FilterMgr Registry table (see Appendix A).
-512	Mux database error when using EDL Editor. Unable to open the mux database table.	Could be an ODBC error. Run mdac_type.exe provided with current installation. Check database using MS Access.

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-513	IBM-Video database error when using EDL Editor.	Could be an ODBC error. Run mdac_type.exe provided with current installation
-514	IBM-Audio database error when using EDL Editor.	Could be an ODBC error. Run mdac_type.exe provided with current installation.
-515	Storage database error when using EDL Editor.	Could be an ODBC error. Run mdac_type.exe provided with current installation.
-516	VTR database error when using EDL Editor.	Could be an ODBC error. Run mdac_type.exe provided with current installation.
-517	Invalid DSN supplied when using EDL Editor.	Could be an ODBC error. Run mdac_type.exe provided with current installation. Also, check EDL Editor properties to be sure that it is associated with the correct DSN ("BroadcastArgus") and be sure that "BroadcastArgus" is registered on the system as a database source.
-518	Attempted to access the ODBC load or save when not in EDL Editor mode.	
-519	Attempted to schedule pause/resume when dual-encoding turned on. Not allowed.	
-520 … -603	Multi-stream encode error.	User should not attempt multi-stream encodes with Argus 4:2:2/4:2:0 software.
-610	Argus VTR configuration error. Attempted to turn on VTR control when Registry indicated that VTR is not installed.	Check EncodeConfig Registry. Turn on VTRInstalled flag if you intend to use the VTR.
-611	Unable to create mutexes for video component.	System error? Check task manager.

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-612	Unable to create IBM video COM component.	Make sure IBMVideo component is registered.
-613	Exception thrown during IBM Video initialization.	
-614	Error occurred during IBM Video stop procedure.	This may result in (or may have resulted from) “hung up” encode. May need to terminate it.
-615	Error occurred with IBM-video pause command.	
-616	Error occurred with IBM-video resume command.	
-617	Unable to create mutexes for IBM Audio component.	
-618	Error occurred during IBM Audio stop.	
-619	Error occurred with IBM Audio Pause.	
-620	Error occurred with IBM Audio Resume.	
-621	Error occurred with IBM Audio Reset.	
-622	Failure communicating with VSP hardware.	Check seating of encoder board. Run diagnostics.
-623	Failure mapping VSP.	Check seating of encoder board. Run diagnostics.
-624	VSP driver “open” command failed.	Check seating of encoder board. Run diagnostics.
-625	VSP reset failed. Unable to reset encoder.	
-626	VSP unmap failed.	
-627	Unable to create mutexes for FTP component	
-628	Unable to create FTP COM object.	Make certain that the RemoteStore component is registered.
-629	Error initializing RemoteStore component.	

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-630	Error resetting RemoteStore component.	
-700 … -726	Multi-stream encode error.	User should not attempt multi-stream encodes with Argus 4:2:2/4:2:0 software.
-727	Mux failed to start.	Usually the result of an audio or video board failure.
-728	Video input ended.	Video input stopped unexpectedly. Check for PAL/ NTSC or analog/digital conflict. Run diagnostics.
-729, -730	Audio input ended.	Audio input stopped unexpectedly. Check for PAL/ NTSC or analog/digital conflict. Run diagnostics
-731	Mux outpin error.	The application was unable to send the muxed stream to the output pin. Check CPU/memory usage.
-732	Invalid VBR bit rate.	The average VBR bit rate entered must be less than the maximum bit rate.
-733	VBR not installed.	The application attempted to run a VBR encode when the VBR microcode has not been installed.
-734	Invalid Mux mark-in.	The mark-in entered for the Mux time-code adjustment is invalid.
-735	Error opening video board. Could be attempting to open a board that is not installed.	Check installation of video boards.
-736	Video unmap error. An error unmapping video-board memory.	Programming error.

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-737	No video data detected. During an attempted encode (usually at the beginning), the video board failed to detect video source data.	Make certain that a video source is connected to the encoder board. If the source type is specified as digital, make certain that there is digital input. Make certain that the PAL / NTSC setting in the video Registry table for the specified board is correct.
-738	Video FIFO overflow.	Application is not emptying the video encoder FIFO (in other words, is not reading the data from the video board) fast enough.
-739, -740	Video FIFO underflow.	Application is attempting to read data from the video encoder board, but none is available. See -737.
-741	Error memory-mapping the video encoder board.	Check video board installation. Run diags.
-743	Cannot use Mux component— It is busy.	Ascertain that previous encode was shut down correctly.
-744	Invalid number of audio pins. An attempt was made to reference an out-of-range audio pin index.	Programming error.
-800	User cancelled encode after cueing, but before starting.	Shut down and restart the application before attempting another encode.
-2001	Encoder board or audio stream index out-of-range.	Registry programming error.
-2002	Audio index out of range. An attempt was made to save audio properties to a Registry table for an undefined board.	Registry programming error.

Table B-1. Filter Manager Error/Status Codes (Continued)

Filter Manager Error/Status Codes (Continued)		
Error Code	**Meaning**	**Comments**
-2003	Video index out of range. An attempt was made to save video properties to a Registry table for an undefined board.	Registry programming error.
-2004	Mux index out of range. An attempt was made to save mux properties to a Registry table for an undefined board.	Registry programming error.
-2005	Storage index out of range. An attempt was made to save file-store properties to a Registry table for an undefined board.	Registry programming error.
-2006	Stream-id conflict. An attempt was made to assign the same stream id to two audio or to two video streams within the same encoded file. Check the stream-id values in the Mux properties table.	Registry error.
-2007	PID conflict. An attempt was made to assign duplicate PID values to two streams within a single transport stream. Check the audio and video PID values in the Mux properties table.	Registry error.
-2008	Invalid time code type. The high-order digit of the GopTcStart field in the Mux Registry table was assigned a value other than 0 (PAL), 1 (NTSC, non-drop-frame), or 2 (NTSC, drop-frame).	Registry error.

Table B-1. Filter Manager Error/Status Codes (Continued)

Index