High Definition Audio for the Digital Home

Proven Techniques for Getting It Right the First Time

David Roach, Scott Janus, and Wayne Jones

Updated to include version 1.0a spec, audio support for Windows 7 and Mac OS† X, plus Windows Hardware Quality Labs (WHQL) logo testing.

Intel PRESS

Books by Engineers, for Engineers

intel®

High Definition Audio for the Digital Home

Proven Techniques for Getting It Right the First time

Updated to include version 1.0a spec, audio support for Windows 7 and Mac OS† X, plus Windows Hardware Quality Labs (WHQL) logo testing.

David Roach
Scott Janus
Wayne Jones

Intel
PRESS

ISBN 1-934053-30-9

This book is printed on acid-free paper. ♾

Publisher: Richard Bowles
Program Manager: Stuart Douglas
Text Design & Composition: Wayne Jones
Graphic Art: Wayne Jones (illustrations), Wayne Jones and Ted Cyrek (cover)

Library of Congress Cataloging in Publication Data:

10 9 8 7 6 5 4 3 2 1

Updated June 2012
Printed in the United States of America

To Phil Pompa and Rich Bowles, who came up with the concept and put the wheels in motion to make it happen

Contents

Preface

Chapter 4 Introduction to Intel High Definition Audio

- Hardware 2
- Controller 3
- Link 4
  - Topology 5
  - Data and Timings 7
- Codec 8
  - Introducing…the Widget 9
  - Widgets in Practice 14
- Command and Data Flow 16
  - Verbs 16
  - Responses 19
- Hardware Volume Scaling 20
  - Pin Widget Control 24
  - Standard Packaging 25
  - Pins, Pin Widgets, and Ports 26
  - Verb Tables 29
  - Sending Verb Tables to the Codec 31
  - Muting and Startup Work-arounds 32
  - Pin Configuration Registers 35
  - Associations and Sequences 42
  - Resource Sharing 48
  - Resource Allocation 54
  - Unsolicited Response 55
  - Jack Detection 55
  - Modifying the Pin Configuration in the BIOS 56

System Bring-up Trick Using the Microsoft UAA class driver for Intel® HD Audio 57
Display-based Audio 58
Data Island Packets 60
HDCP 61
Low Power Enhancements 63

Chapter 12 Windows† 7 and Mac OS† X

Mac OS X and HD Audio 65
Extending the Windows 7 HD Audio Class Driver 66
How Custom APOs Interact with System APOs 68
Using a Custom INF File with the Class Driver 70
Replacing the Enhancements Tab 71
New Audio Communications Features in Windows 7 72
Follow-Me Endpoint Heuristics 73
Dynamic Stream Redirection and Dynamic Format Change 73
Updated Volume Control Experience 74
Capture Monitor 74
Multi-Channel HDMI and DisplayPort support 75
Exclusive Mode 76
Low Power Enhancements 79
Virtualized 8253 Timer 79
System-wide Attenuation or Ducking 80
Event-Driven WaveRT 81
HDCP and SCMS Copy Protection Support 81
Windows Media Player OCX Device Selection 81
Windows Performance Analysis Tools 81
Echo Cancellation and Microphone Arrays 83
HDAU - HD Audio Tool 85
Windows 7 Driver Installation 88
How to Install/Remove Drivers Using Device Manager 88
How to Install/Remove Drivers Using Sound Control Panel 89
Driver Install Frameworks 89
Using SetupAPI.Device.Log to Debug Driver Installations 90
Popular Audio Registry Key Locations 91

Chapter 13 Windows® Logo Testing for HD Audio

Microsoft Windows Hardware Quality Lab (WHQL) Logo Process 93
- The WLK Includes the Driver Test Manager (DTM) 94
- Windows Quality Online Services (WinQual) 95

Driver Submission Process—What Files Are Submitted 95
- WLK Errata 96
- Third-Party Audio Fidelity Testing 97
- Test Signing 97
- Audio Fidelity Requirements 97
- In-Box Class Drivers 98
- Test Duration 98

DTM System 98
- Components of a DTM System 100

What Tests Are Run: Current Audio Fidelity Testing 102

Starting DTM Controller 104

DTM Studio Job Monitor 105
- Creating a Machine Pool 106
- Adding systems to the pool 106
- What to do when you can't get a heartbeat 106
- Choosing Tests to Run 107
- Running Tests 108
- Viewing errors, using DTM console or Log Viewer 111

Running Tests Outside the DTM "Shell" 112
- How to Locate Tests that Run from Outside the Shell 113

Failures 113
- Jacks Not Populated 113
- Noise Failures 114

Summary 114

Preface

The first edition of this book was published in 2006. In the five years since then, both Windows† Vista and Windows 7 have been released, and Intel® High Definition Audio (Intel® HD Audio) has become the standard built-in audio solution for both Macintosh† and Windows-based computers.

This update to the book consists of three chapters. We simply added new material to Chapter 4 covering recent updates to the High Definition Audio Specification version 1.0a, which was released June 17, 2010.

Chapter 12 and Chapter 13 stand on their own and are added to the end of the book. Chapter 12 discusses audio support for Windows 7 and for Mac OS† X, while Chapter 13 goes into detail on how to run the Windows Hardware Quality Labs (WHQL) logo testing that is required for systems that support Windows 7.

Chapter 4

Introduction to Intel® High Definition Audio

Those parts of the system that you can hit with a hammer (not advised) are called hardware; those program instructions that you can only curse at are called software.

—Unknown

This chapter introduces the information needed to understand the hardware components that make up an audio subsystem in a modern PC, after the shift to chipset-based audio. It also addresses the attributes and constraints of several key hardware components such as converters and analog amplifiers. Finally, the Intel® High Definition Audio interface is introduced and explored in some detail.

In 2004, Intel announced an Audio Codec 97 (AC97) replacement known as Intel® High Definition Audio (Intel® HD Audio). Introduced in 1997, the AC97 standard allowed sound-processing tasks such as sample rate conversion and mixing to be handled by the CPU. Logic known as the *AC97 digital controller* was added to the Southbridge or ICH chipset, as shown in Figure 4.1.

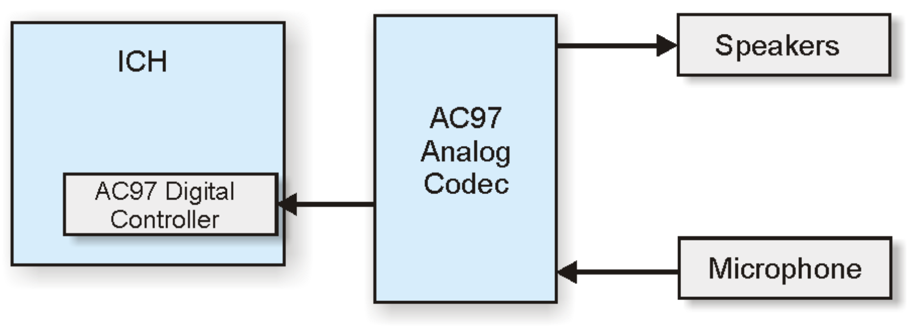


Figure 4.1 Addition of AC97

AC97 was originally known as *soft audio* or *host audio* because it offloaded many of the sound-processing tasks from dedicated hardware to software that is run on the general purpose CPU. Intel HD Audio is thematically similar to AC97 but includes support for many new features. For instance, Intel HD Audio can support 32-bit, 192-kilohertz surround sound out as well as multitasking, which is the ability to support multiple audio inputs and outputs simultaneously.

Quite a bit of the AC97 infrastructure is retained for Intel HD Audio. It uses the same five-wire serial link between controller and codec, as well as the same 48-pin package for the audio codec. Much of the analog circuitry in the codec remains the same, though better Intel HD Audio codecs have improved the audio fidelity of these circuits. While the Intel HD Audio specification allows for up to 16 input and output streams; today's chipsets typically support four input and four output streams. A separate DMA engine is required for each stream.

Hardware

The three key types of physical components in an Intel HD Audio solution are an Intel HD Audio Controller, an Intel HD Audio Link, and an Intel HD Audio Codec, as shown in Figure 4.2. The controller has responsibility for moving data to and from system memory via a memory controller. The link is the bi-directional interconnection across which audio data and commands flow. The codec interprets the commands and sends or receives data to or from an attached acoustic device—that is, a speaker or microphone.

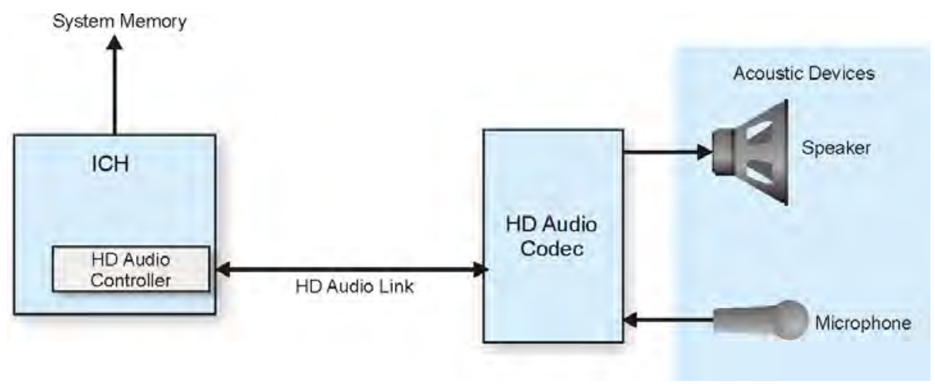


Figure 4.2 Intel® HD Audio Components

Multiple codecs can be attached to a single link. More rarely, a single system could have multiple controllers, each of which would have its own associated link and attached codecs. This situation might occur if a soundcard with an Intel HD Audio controller is inserted into a PCI or PCI Express slot.

Controller

An Intel HD Audio controller provides a capability for the system to discover and enumerate multiple Intel HD Audio devices connected to the Intel HD Audio link. It provides a capability to relay instructions to and from each codec, and it contains the Direct Memory Access (DMA) engines that stream audio data to and from the codec(s).

An Intel HD Audio Controller usually contains four or more DMA engines. DMA refers to copying data from one memory location to another without having to go through the CPU. In this case, the transfer goes from system memory to a codec or goes from a codec to system memory (see Figure 4.3). DMA transfers are faster than non-DMA transfers.

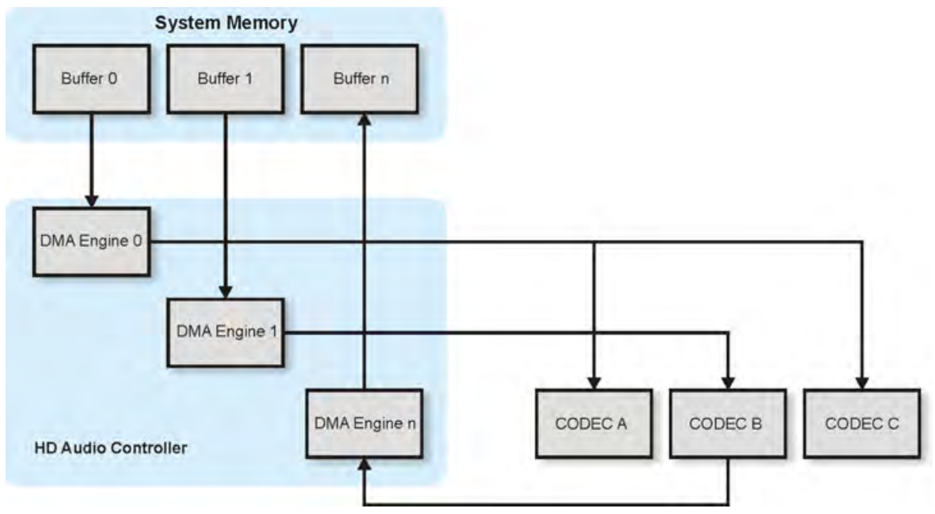


Figure 4.3 Controller DMA Engines

A DMA transfer from system memory to a codec is handled by a signal known as Serial Digital Out (SDO). DMA transfers in the opposite direction are handled by a Serial Digital In (SDI) signal. Controllers must provide support for at least one SDO line, up to a maximum of four, and at least one SDI line, up to a maximum of fifteen. The SDO and SDI lines are used to carry both commands and audio sample data, as explained in more detail in the "Command and Data Flow" section. In addition to the SDO and SDI lines, the controller must support bit clock (BCLK), frame synchronization (SYNC), and reset (RST#) signals.

The controller also contains a standardized set of memory-mapped registers that allow for command and control of the data streams and the codecs. The standardized nature of a controller's register makes it straightforward to write a universal Intel HD Audio Controller software driver that works for multiple instantiations of controllers, even if they are produced by different manufacturers, something Microsoft is utilizing for their Windows class driver initiative, Universal Audio Architecture.

Link

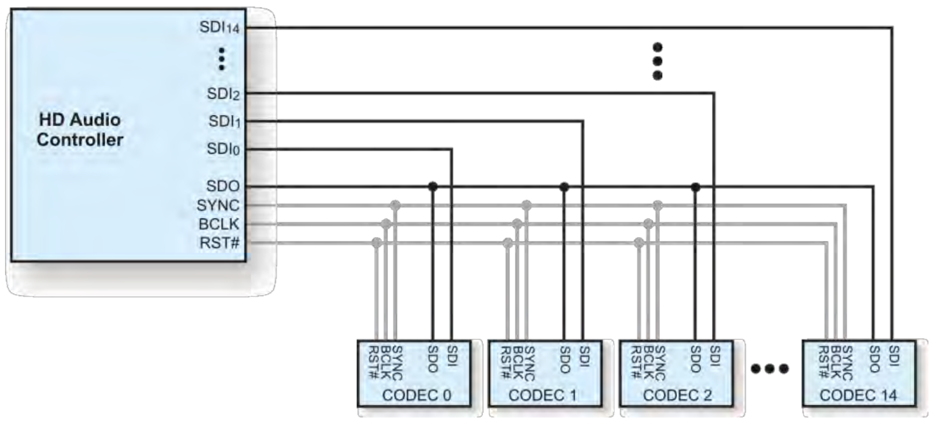
The link itself, which connects a controller with one or more codecs, is a physical electrical interface consisting of the BCLK, SYNC, and RST#

signals. It also consists of one to four SDO signals, denoted SDO_0-SDO_3 respectively, and one to fifteen SDI signals, and denoted SDI_0-SDI_N respectively.

Topology

The SDO lines can be multipoint; that is, a single SDO line can be attached to multiple codecs. In a configuration like the one in Figure 4.4, multiple codecs could receive the same audio data, thus allowing, for instance, a set of headphones and separate stereo speakers to receive the same program. However, the same physical topology could be used to send different programs to the headphones and speakers. The data sent from the controller can be targeted at a specific codec.

Unlike SDO, the SDI lines are point-to-point. Each attached codec's SDI must be wired to a unique SDI_n on the controller. Given that each codec must have at least one SDI, a practical upshot of these constraints is that the maximum number of codecs that can be attached to a particular link is limited by the number of SDIs the controller supports.

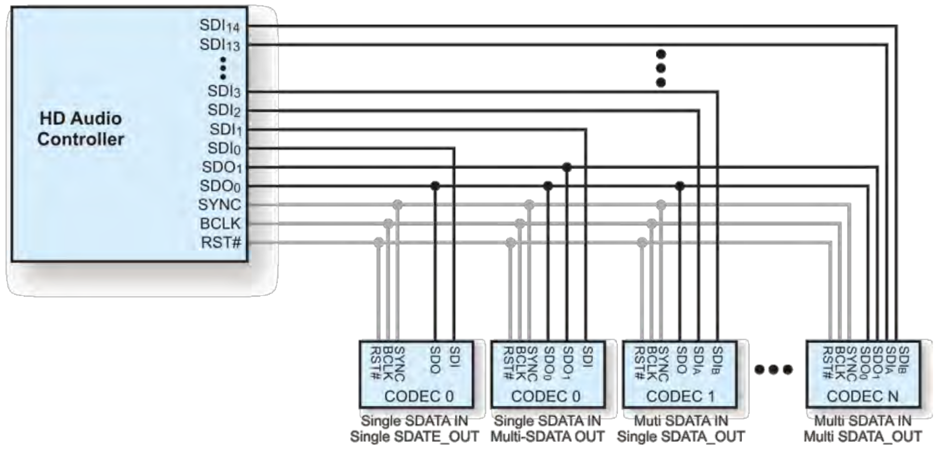


In this example, the controller's single SDO is wired to every codec's SDO. However, every SDI has a unique path—a fundamental requirement of the specification. More than one SDI line is optional for any particular system.

Figure 4.4 Sample Link Topology

Most, if not all, Southbridge chipsets that incorporate Intel HD Audio have at least four SDI pins, allowing up to four Intel HD Audio devices to be used in a system. The SDI pin that the codec is attached to determines the Codec ID used to address a specific codec. SDI0 is attached to Codec ID 0, SDI1 is attached to Codec ID 1, and so on. Intel recommends that SDI#2 be used for codecs mounted on the motherboard, and that SDI#0 be used for a modem codec, if present. SDI#1 and SDI#3 are reserved for future usage models.

To allow for future expansion, both the output and input data rates for Intel HD Audio are scalable. A single SDO or SDI provides the basic throughput, but *bandwidth scaling* is achieved by attaching multiple SDO or SDI lines to a codec (see Figure 4.5). The data bandwidth is essentially increased by the number of lines. For instance, four SDO lines offer roughly four times the bandwidth of a single SDO line. While technically feasible under the specification, no OS currently supports bandwidth scaling.



In this example, all four types of bandwidth-scaling codecs are attached to a link. Although the multiple in/out codecs only show two associated lines, it is possible to build systems with larger numbers of connections.

Figure 4.5 Bandwidth Scaling

With bandwidth scaling in mind, codecs can be classified into four categories:

- Single-SDO, single SDI
- Multi-SDO, single SDI
- Single-SDO, multi-SDI
- Multi-SDO, multi-SDI

Even in the case of a multi-SDI codec, the SDI lines remain point-to-point. Each SDI on the codec must be attached to a unique SDI on the controller.

In all systems, even those using bandwidth scaling, SDO_0 must be attached to all codecs on the link. This restriction ensures the basic operation of the system in the scenario of codecs that support scaling running with software that does not. The number of permitted SDO lines is 1, 2, or 4.

When building a system using codecs that are multi-SDO or multi-SDI, it is very important to attach all of the SDO and SDI lines from the codec to the controller. Failure to do so could result in reduced codec performance.

Data and Timings

The Intel HD Audio Link is isochronous, meaning that it can provide data at a predictable and fixed rate, rather than in bursts like most networks. The isochronicity is achieved in part from the use of a fixed bus clock (BCLK) that runs at 24.000 megahertz. Data transfers on the bus are broken into frames, each of which is exactly 20.833 microseconds. The frames are designated by the frame sync (SYNC) signal's falling edge, shown in Figure 4.6. Each frame ends with a SYNC signal going high for exactly four BCLK cycles, an event known as the frame sync.

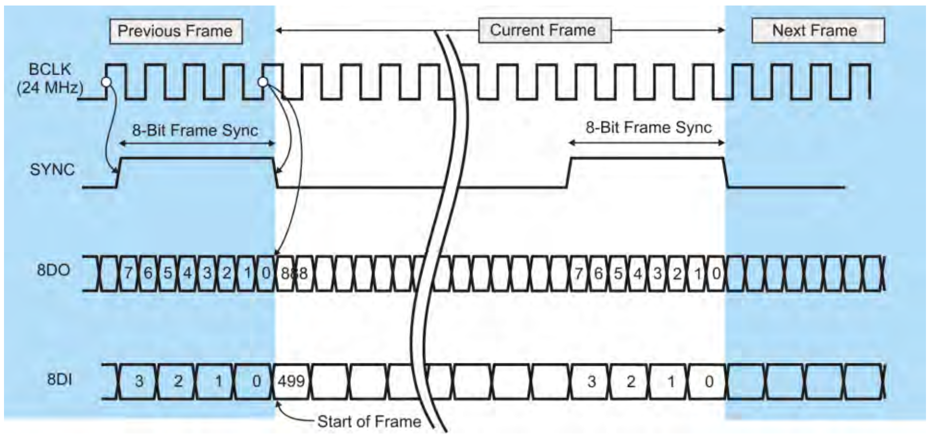


Figure 4.6 Link Signals

SDO is *double-pumped*, meaning that a unique bit of data can be sent on both the rising and falling edge of the reference clock (BLCK). Given that BLCK runs at 24 megahertz, a single SDO signal can convey 48 megabits per second. Each bit is contained within a timing window known as a *cell*. For SDO, each frame consists of 1,000 cells.

SDI is single-pumped. Only one bit is transferred per clock cycle. As such, SDI has half the bandwidth of SDO, or 24 megabits per second. Similarly, SDI has only 500 cells per frame.

The data within a frame can contain commands, status, and audio samples. The detailed structure of a frame will be examined more closely in the "Frame Format" section in this chapter.

Codec

Intel HD Audio Codecs are modular. They consist of a hierarchy of standardized modules. The number and types of modules, plus their connectivity, may vary from one codec to another. The codec architecture includes a discovery and addressing scheme that allows for a single driver to easily support a wide variety of codecs.

A *node* is either a single module within a codec or it is a collection of a module and all its children modules that are connected below it in the hierarchy, as shown in Figure 4.7. Each node has a unique address, known as a *node ID* (NID). An NID is usually 7 bits, but 15-bit NIDs can be used for very complex systems. Each node has a set of read-only capabilities, and each can be controlled and configured using commands targeted at that node.

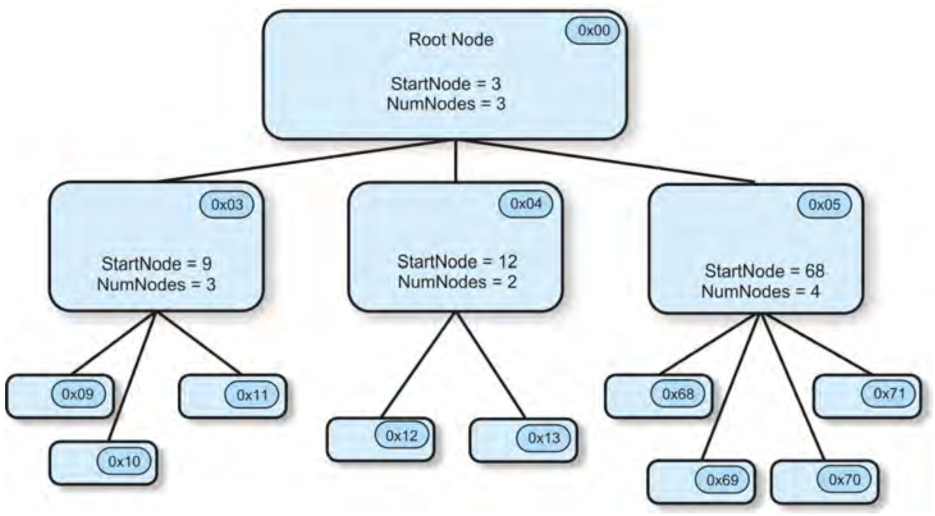


Figure 4.7 Codec Node Hierarchy

The *root node* is the node at the top of the hierarchy. It has an NID of zero. The root node has no other function than to serve as a pointer to the downstream nodes. The root node points to one or more *function group nodes*. A function group node is a collection of modules that perform a dedicated function and are designed to be controlled by a single driver.

The two basic function group types in use today are audio and modem; other function groups may be defined in the future. The remainder of this chapter focuses on the audio function group (AFG). The modem function group is outside the scope of this book. Some newer audio codecs support both audio and modem function groups in the same die, but each has it's own SDI pin on the codec, and they are treated as totally separate units.

Introducing…the Widget

Below each function group node is a collection of modules known as *widget nodes* or *widgets*. Widgets can be interconnected in many different ways, allowing a single AFG to support an arbitrary number of audio input and output channels.

A widget whose inputs are connected to the outputs of other widgets must contain a *connection list* that consists of the NIDs of the attached widgets. The connection list refers to the hard-wired topology of the interconnected widgets. If the connection list length is greater than one, the widget would have a *connection selector* that allows the runtime selection of input(s) to be used by the widget.

Thus, connection lists are always built from right to left in terms of conventional left-to-right audio flow. For output devices, you start with the output port and follow the connection list towards the Intel HD Audio bus. For input devices, you start with the ADC where it is attached to the Intel HD Audio bus and follow the connection lists widget by widget until you reach an input port.

Widgets are either 1-channel (mono) or 2-channel (stereo). An AFG as a whole can support greater than 2-channel sound by using multiple widgets.

The types of standardized audio widgets are:

- Audio Output Converter (AOC) Widget
- Audio Input Converter (AIC) Widget
- Pin Widget
- Mixer Widget
- Selector Widget
- Power Widget
- Volume Knob Widget
- Beep Generator Widget

An Audio Output Converter Widget converts audio data coming in over the link into an analog or digital audio format more suitable for transmission to an acoustic device. The key component of an AOC widget is a DAC for analog outputs or a digital formatter for digital outputs, such as S/PDIF. The widget may contain either an optional processing node or an optional amplifier with a mute capability or both, as shown in Figure 4.8. The data input to an AOC widget is always attached directly to the link.

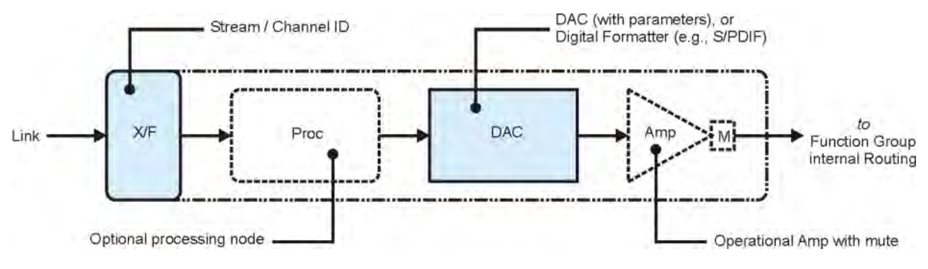


Figure 4.8 Audio Output Converter Widget

As you might guess, the Audio Input Converter Widget is the converse of the AOC. It converts analog or digital external formats into a digital format compatible with the link. As such, the main component of an AIC is an ADC or a digital sample formatter. An AIC may also contain an optional processing node and/or an optional amplifier with mute capability (Figure 4.9). The data output of the AIC attaches directly to the link.

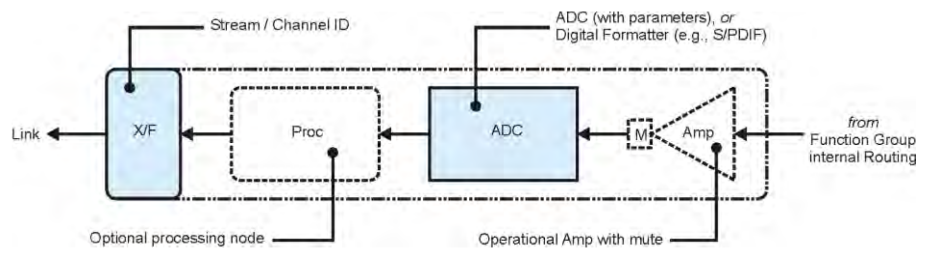


Figure 4.9 Audio Input Converter Widget

The Pin Widget shown in Figure 4.17 is a mechanism for sending to or receiving audio signals from a physical audio connector or sending them to a hard-wired connection to a dedicated acoustic device, such as a built-in speaker or a microphone integrated in a laptop bezel. The Pin Widget also supports other signals that are associated with the connector, such as *jack sense*—it detects the presence and type of the acoustic device that is plugged into the connector and VRefOut —used to switch microphone bias voltage on and off. The Pin Widget may contain optional outgoing and ingoing amplifiers. Alternatively, the Pin Widget is referred to as the Pin Complex Widget.

The Pin Widget is also very important because it can be used to convey detailed information about the associated connector. For instance, the type of connector, its location on the PC, and the color of the connector can all be communicated by the pin widget to the software stack. When such information is made available, very user-friendly interfaces can be designed which show computer users on-screen what connections are active or available on their computer. See "Pins, Pin Widgets, and Ports" later in this chapter for more detail on pin configurations.

The Mixer Widget, also known as a Summing Amplifier Widget, is used to combine multiple signals into a single signal (see Figure 4.10). The number of inputs can vary. The relative intensity of the different inputs can also be altered.

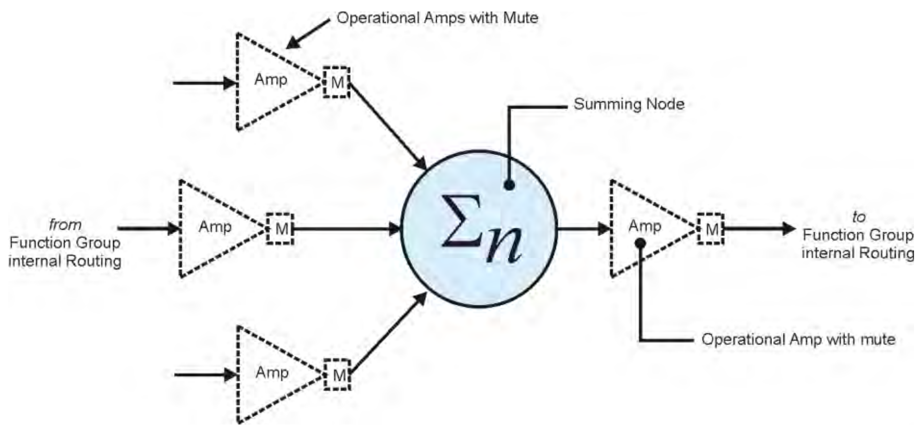


Figure 4.10 Mixer Widget

The Selector Widget is used to select a single signal from a multitude, as can be seen in Figure 4.11. Sometimes, the Selector Widget is referred to as a multiplexer widget (Mux for short). As it passes through the Selector Widget, the signal can be processed, amplified, or muted, if such optional capabilities are supported by the widget.

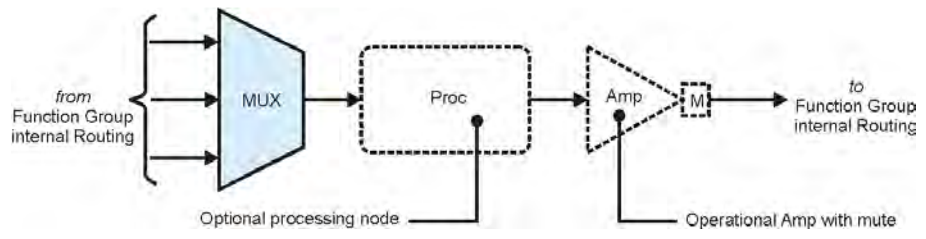


Figure 4.11 Selector Widget

Since most widgets already have implicit selection mechanisms via their connection lists and connection selector controls, the Selector Widget is not needed in most designs and has rarely appeared in actual codecs. However, a 1-input selector could be used to insert a processing/amplifier widget arbitrarily into a widget hierarchy.

The Power Widget provides a single point of control for power management of a subset of the widgets in an Audio Function Group, independent of the power management of the AFG as a whole. The Power Widget's connection list contains a static list of all the widgets under its control. The power state of those widgets can then be set independently of the AFG's power state, with the constraint that the Power Widget can never be at a higher power state than the AFG.

The Volume Knob Widget is used to report the state of a physical volume control. The control could either be absolute, as would be the case for a thumbwheel potentiometer, or it could be relative, as would be the case for + and – pushbuttons. The *Master Volume in Windows*† *XP* section of Chapter 8 includes details on how the Volume Knob Widget can be used with the Windows operating system.

The Beep Generator Widget is used to…generate a beep! The beep is derived from the 48-kilohertz reference clock on the link. A variety of frequencies can be created by specifying a divisor to be applied to the 48-kilohertz reference. The Beep Widget is not explicitly attached to other widgets via connection lists. Indeed, a Beep Widget is forbidden from appearing on the connection lists of other widgets. Instead, when the Beep Widget is activated, the tone that it generates must be applied to all active output pin widgets. The beep can either replace or be blended with other signals that might already be active on those pins.

Widgets in Practice

From the eight simple building blocks listed above, complex codecs can be readily constructed. Figure 4.12 shows an example of the widgets that are used in a real-world codec. This codec supports S/PDIF in and out, eight channels of analog out, multiple stereo inputs, I^2S out, ADAT out, and PC Beep.

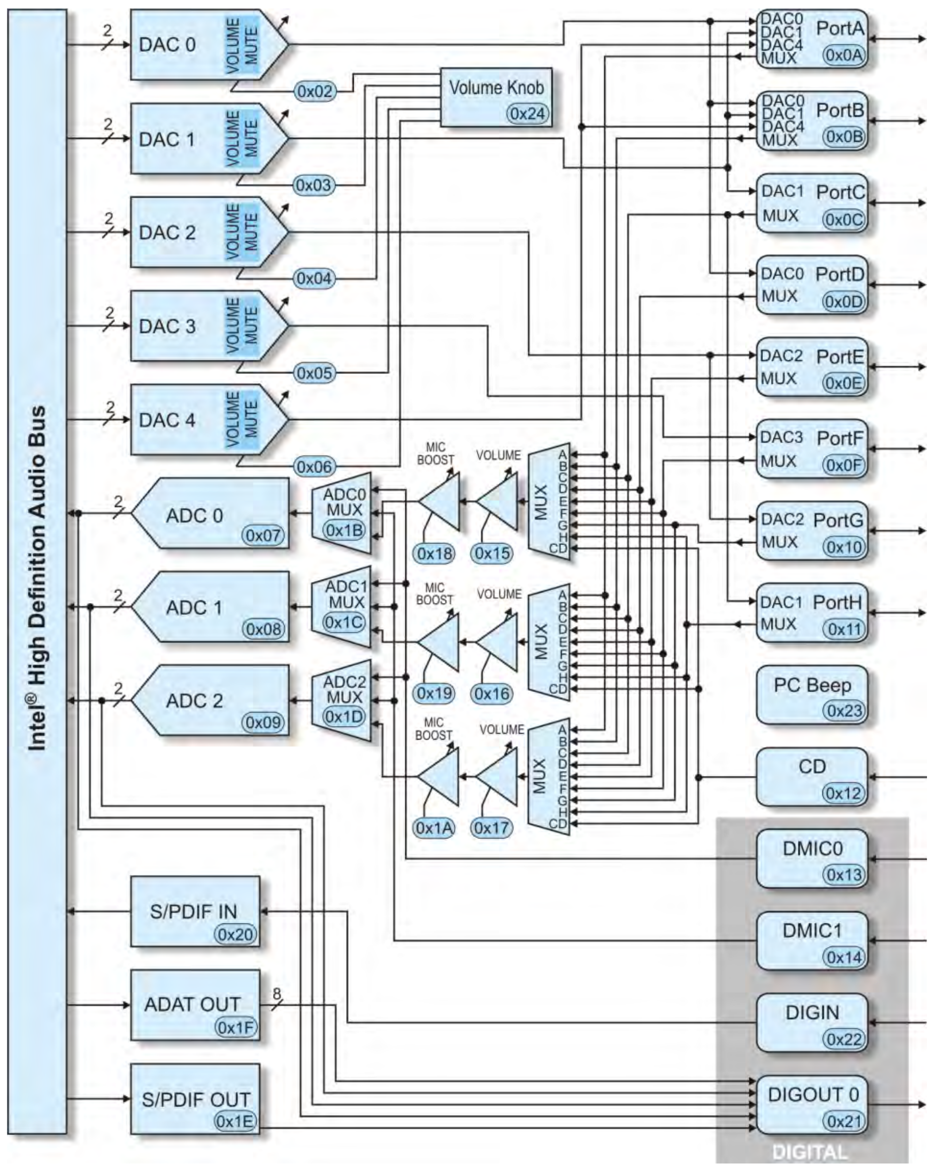


Figure 4.12 Sample Widget Diagram

Command and Data Flow

As mentioned, SDO and SDI carry both commands and audio data. The commands and their associated responses are low-bandwidth; the audio data is high-bandwidth. The two forms of data are interleaved, but the outbound protocol is slightly different than the inbound protocol.

To understand how these forms are multiplexed together, it is important to first understand the concept of a stream. A *stream* is a connection between a DMA buffer in the controller and a codec. A stream contains one or more audio channels from the same audio program. For instance, a stereo program would contain two channels.

Streams can either be output streams conveyed over SDO or input streams conveyed over SDI. Given the multipoint topology of SDO, it is not surprising to learn that output streams can be broadcast to multiple codecs. For instance, the same audio stream can be rendered concurrently on both headphones and speakers.

Verbs

These streams transport audio samples in both directions across the link. Although the audio samples take up the majority of the link bandwidth, they are useless without ancillary control data that is used to modify and direct the flow of the audio samples. The command information originates from the controller, which issues 32-bit commands known as verbs. Figure 4.13 shows the structure of verbs.

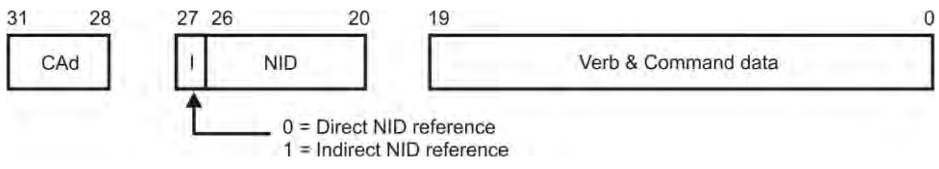


Figure 4.13 Verb Structure

The verb contains a 4-bit codec address that indicates the codec to which the command is targeted. A value of all ones in this field (`Fh`) is reserved to verbs that are intended to be broadcast to all codecs on the link. However, no such broadcast verbs have been defined in the current revision of the specification.

The verb also contains an 8-bit node address, which indicates the node at which the verb is targeted. Two different node-addressing schemes are supported: direct and indirect. If bit 27 of the verb is 0, the node ID (NID) is a direct address. Otherwise, it is an indirect address. However, an indirect addressing scheme has not yet been specified, so in practice, all verbs use direct addresses. The 7-bits allow a single codec to contain 127 nodes.

The final 20 bits of the verb command the actual command and its parameters. Verbs either have a 4-bit command identifier with a 16-bit payload or a 12-bit command identifier with an 8-bit payload. Not all types of widgets need to support all verbs. Table 4.1 is a summary of the verbs and whether or not they are required for the different widgets. Many verbs have get and set variations.

Table 4.1 Required Support for Verbs

Required Verb Support	**Get Code**	**Set Code**	**Root Node**	**Audio Function Group**	**Modem Function Group**	**Vendor Defined Function Group**	**Audio Output Converter**	**Audio Input Converter**	**Pin Complex Widget (non Digital Display)**	**Pin Complex Widget (Digital Display)**	**Mixer**	**Selector**	**Power Widget**	**Volume Knob**	**Beep Generator**	**Vendor Defined Wdiget**
GetParameter	F00		R	R	R	R	R	R	R	R	R	R	R	R	R	R
ConnectionSelect	F01	701						C	C	C		C				
GetConnectionList Entry	F02							R	R	R	R	R	R	R		
Processing State	F03	##					C	C	C	C		C				
Coefficient Index	D	5					C	C	C	C		C				
Processing Coefficient	C	4					C	C	C	C		C				
Amplifier Gain/Mute	B	3					C	C	C	C	C	C				
Stream Format	A	2					R	R								
Digital Converter 1	F0 D	70 D					C	C								
Digital Converter 2	F0 D	70 E					C	C								
Digital Converter 3	F0 D	73 E					C	C								
Digital Converter 4	F0	70F					C	C								

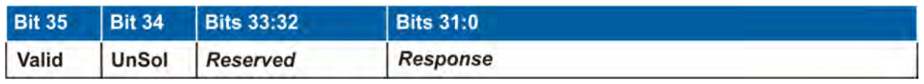
	D															
Power State	F05	705		R	R	C	C	C	C	C	C	C	R			
Channel/Stream ID	F06	706					R	R								
SDI Select	F04	704					X	X								
Pin Widget Control	F07	707							R	R						
Unsolicited Enable	F08	708					C	C	C	C	C	C	C	C		
Pin Sense	F09	709							C	C						
EAPD/BTL Enable	F0 C	70 C							C							
All GPI Controls	F10 – F1 A	710 – 71 A		C	C											
Beep Generation Control	F0 A	70 A													R	
VolumeKnobContr ol	F0F	70F												R		
SubsystemID,Byte 0	F20	720		R	R	R										
SubsystemID,Byte 1	F20	721		R	R	R										
SubsystemID,Byte 2	F20	722		R	R	R										
SubsystemID,Byte 3	F20	723		R	R	R										
ConfigDefault,Byte 0	F1 C	71 C							R	R						
ConfigDefault,Byte 1	F1 C	71 D							R	R						
ConfigDefault,Byte 2	F1 C	71 E							R	R						
ConfigDefault,Byte 3	F1 C	71F							R	R						
Stripe Control	F24	724					C									
Converter Channel Count	F2 D	72 D					C									
DIP-Size	F2 E									R						

ELD Data	F2F									R						
DIP-Index	F30	730								R						
DIP-Data	F31	731								R						
DIP-XmitCtrl	F32	732								R						
Content Protection Control	F33	733								C						
ASP Channel Mapping	F34	734								R						
RESET		7FF		R	R	R										

R: Required.
C: conditional; required only if respective optional capability is declared to be available
X: required if codec supports multiple SDI signals.

Responses

Verbs are synchronous commands. For every verb that is issued from a controller to a codec, that codec must generate a 36-bit response on the following frame. The format of the response is shown in Figure 4.14. Bit 35 is the Valid bit. It must be set to 1 in order for the controller to process the response.



Bit 35	Bit 34	Bits 33:32	Bits 31:0
Valid	UnSol	*Reserved*	*Response*

Figure 4.14 Response Structure

The lower 32 bits of a response is a data payload. Responses to Set commands typically have a payload of all zeros, whereas responses to Get commands typically contain the information requested by the verb.

Bit 34 is used to flag a special type of response known as an *Unsolicited Response.* Unsolicited responses are asynchronous messages sent from the codec to the controller to signal an event, such as the attachment of an acoustic device. No verb is involved in the transmission of an unsolicited response.

Frame Format

Each frame contains a verb or response, one or more stream packets, and a null field. The exact formats of inbound and outbound frames differ slightly, but conceptually, both types of frames are organized as shown in Figure 4.15.

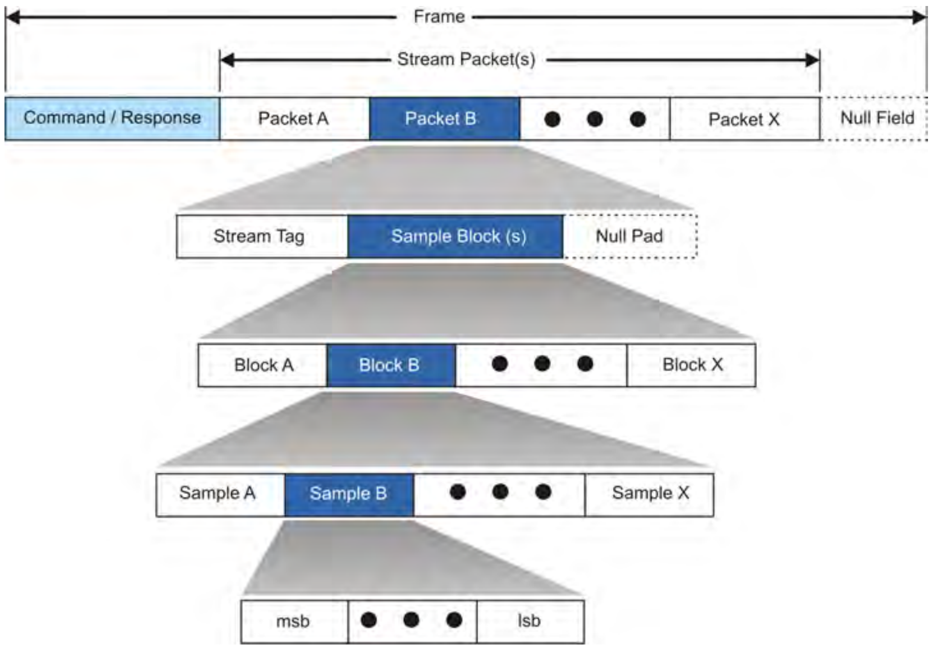


Figure 4.15 Frame Format

The stream packets are logical containers for the audio data streams. Each packet begins with 8-bit stream tag that uniquely identifies the streams. Codecs are programmed by software to send or receive a stream with the specified ID—for example, via the SetConverterStreamChannel verb. After the stream tag, the packet contains one or more sample blocks. Each block in turn contains audio sample data.

Hardware Volume Scaling

Volume controls on Intel HD Audio codecs can be present on almost any widget in the codec. Volume controls are optional and not required. A driver might or might not choose to expose any particular volume control, especially if the signal path contains multiple volume controls. Volume controls in Intel HD Audio codecs are required to default to unity gain, with neither gain nor attenuation. Therefore, volume controls that aren't manipulated by the driver pass through any audio in the signal path. You have no guarantee that the audio engine on the system will make use of the hardware volume controls. Many Intel HD Audio designs lock the hardware volume controls to unity gain, and they perform all volume and muting in software.

Almost every widget in the codec uses the Amplifier Capabilities response format shown in Table 4.2 to report its volume control characteristics. A widget that is retaskable may include both input and output volume controls, each with a separate set of capabilities. You must know the values in StepSize, NumSteps, and Offset for each widget in order to set the volume control correctly. You might need to read this information directly from the codec, as it is often not included in the codec datasheet.

Table 4.2 **32-bit** Amplifier Capabilities Response Format

31	30:23	22:16	15	14:8	7	6:0
Mute Capable	Rsvd	StepSize	Rsvd	NumSteps	Rsvd	Offset

The Amplifier Capabilities response formats are:

- Mute Capable (1 bit) reports whether the respective amplifier is capable of muting. Muting implies a minus infinity ($-\infty$) gain, so that no sound passes, but the actual performance is determined by the Intel HD Audio codec.
- StepSize (7 bits) indicates the size of each step in the gain range. Each individual step may be 0-32 decibels specified in 0.25-dB steps. A value of `0` indicates 0.25-dB steps, while a value of `127d` indicates 32-dB steps.
- NumSteps (7 bits) indicates the number of steps in the gain range. The number could be from 1 to 128 steps in the amplifier gain range. (`0d` means 1 step, `127d` means 128 steps). A value of `0d` (1 step) means that the gain is fixed and may not be changed. Because of this behavior, the name actually is incorrect; technically the name should really be "number of steps minus one."
- Offset (7 bits) indicates which step is 0 decibels or unity gain. With two or more steps, one of the step values must correspond to a value of 0 dB. The Offset value indicates the value which, if

programmed in to the Amplifier Gain control, would result in a gain of 0 decibels.

For example, the AC97 specification defined master volume control as 32 steps of attenuation at 1.5 dB, where the maximum setting was unity gain. This definition is shown on the first row of Table 4.3. The next row shows a more capable volume control with a range of -96 to 0 dB in 0.75 steps. Following that is a volume control from –64 to 0 dB in 0.5 steps. This progression provides smoother steps due to the smaller step size, but the range is limited as a result, since both of these examples use the maximum of 128 steps. The following example shows a typical microphone input slider with a gain from 0 to +22.5 dB, and the final example shows a microphone preamp stage with variable settings of +0, +10, +20, +30, and +40 dB.

Table 4.3 Examples of Volume Controls Defined Using Amplifier Capabilities

Num Steps	Step Size	Offset	True Number of Steps	Lowest Gain	Highest Gain	Step Size	A gain value of 10 equals
31	5	31	32	-42.5 dB	+0.0 dB	1.5 dB	-27.5 dB
127	2	127	128	-96.0 dB	+0.0 dB	0.75 dB	-88.5 dB
127	1	127	128	-64.0 dB	+0.0 dB	0.5 dB	-59 dB
21	3	11	22	-10.0 dB	+10.0 dB	1.0 dB	-1.0 dB
15	5	0	16	+0.0 dB	+22.5 dB	1.5 dB	+15 dB
4	39	0	5	+0.0 dB	+40 dB	10 dB	+40 dB*

Note: The three columns on the left of the double line show the Intel HD Audio parameters, while the five columns on the right show the real world values of each example. The leftmost column shows the value of the NumSteps field, which is really the number of steps minus one. The rightmost column shows the effective gain in dB when a numeric value of 10 is written to the gain register in a widget.

The Set Amplifier Gain/Mute verb supports the 16-bit payload shown in Table 4.4, which is capable of specifying a number of simultaneous parameters for a stereo widget. You can set all of the volume controls in a widget to the same value in a single operation, or you can send multiple verbs to set each input and output control separately on each channel.

Table 4.4 **16-bit Amplifier Gain/Mute Payload**

15	14	13	12	11:8	7	6:0
Set Output Amp	Set Input Amp	Set Left Amp	Set Right Amp	Index	Mute	Gain

The bits in the Set Amplifier Gain/Mute verb are:

- Set Output Amp and Set Input Amp determine whether the value programmed refers to the input amplifier or the output amplifier in widgets which have both, such as pin widgets, sum widgets, and selector Widgets. A value of 1 indicates that the relevant amplifier should accept the value being set. If both bits are set, both amplifiers are set; if neither bit is set, the command is effectively a no-op. Any attempt to set a non-existent amplifier is ignored.
- Set Left Amp and Set Right Amp determine whether the left (channel 0) or right (channel 1) channel of the amplifier is being affected. A value of 1 indicates that the relevant amplifier should accept the value being set. If both bits are set, both amplifiers are set. Any attempt to set a non-existent amplifier is ignored. If the widget only supports a single channel, channel 1 bits have no effect and the value programmed applies to the left (channel 0) amplifier.
- Index is only used when programming the input amplifiers on Selector Widgets and Sum Widgets, where each input may have an individual amplifier. The index corresponds to the input's offset in the Connection List. If the widget being programmed is not a Selector Widget or a Sum Widget, or the Set Input Amp bit is not set, this field is ignored. If the specified index is out of range, no action is taken.
- Mute selects -∞ gain (the lowest possible gain), but the hardware implementation determines the actual degree of mute provided. A value of 1 indicates that the mute is active. Generally, mute should default to 1 on codec reset, but in some circumstances, mute defaults to its off or inactive state. In particular, if an analog PC Beep Pin is used, the mutes of associated outputs must default to 0 to enable the beep signal prior to the codec coming out of reset. This bit is ignored by any amplifier that does not have a mute option.

- Gain is a 7-bit "step" value specifying the amplifier gain, the actual decibel value that is determined by the StepSize, Offset, and NumSteps fields of the Output Amplifier Capabilities parameter for a given amplifier. After codec reset, this Gain field must default to the Offset value, meaning that all amplifiers, by default, are configured to 0 decibels or unity gain. If a value outside the amplifier's range is set, the results are undetermined.

To set a volume control to unity gain, follow these steps:

1. Simply copy the Offset from the Amplifier Capabilities response into the Gain field of the payload.
2. Set the Mute bit to 0.
3. Specify the input or output channel, the left or right channel, and the index (if needed).
4. Then send the payload to the widget.

Without knowing the offset, the only value you can reliably program in the gain field is the number zero (not 0 dB), which always represents the lowest gain available in the amplifier. A payload of `0xF080` mutes both input and output amplifiers on both left and right channels and sets the gain to the lowest possible value.

Pin Widget Control

In addition to the configuration default register, each Pin Widget also has a control register that the audio function driver uses to configure the port. Table 4.5 shows the contents of the Pin Widget Control register.

Table 4.5 Pin Widget Control Register

Bit 7	Bit 6	Bit 5	Bits 4:3	Bits 2:0
H-Phn Enable	Out Enable	In Enable	Reserved	VRefEn

The bits for the Pin Widget Control register are:

- H-Phn Enable stands for Headphone Enable, which disables or enables a low-impedance amplifier to be associated with the output. A value of one enables the amp. Enabling a non-existent headphone amp has no effect.
- Out Enable allows the output path of the port to be shut off. A value of one enables the output path. Enabling a non-existent output has no effect.

- In Enable allows the input path of the port to be shut off. A value of one enables the input path.
- VRefEn stands for Voltage Reference Enable, which controls the switchable microphone bias provided by the VRefOut pin that is associated with the port. A range of settings provide the ability to match specific microphones and to avoid crosstalk when the jack is re-tasked. Adjusting VRefEn sets the microphone bias voltage on the VRefOut pin to one of the following values:
  - Hi-Z (also known as floating, or tri-stated)
  - 100 percent of the analog power supply
  - 80 percent of the analog power supply
  - 50 percent of the analog power supply
  - Ground

Standard Packaging

As should be evident, the Intel HD Audio architecture is designed to allow a wide variety of controllers and codecs from potentially different vendors to operate using a universal software stack. Such a stack might only provide basic functionality. Codec vendors can provide value-add to their products by providing specialized drivers that take advantage of advanced features unique to their codecs.

Continuing this thread of interoperability, the Intel HD Audio specification includes the definition of a 48-pin package for codecs, as shown in Figure 4.16. While this package and pinout is not a requirement, most codec vendors follow it, which allows system integrators to have flexibility in selecting codecs. A motherboard designed to use a 48-pin codec can be populated with any compliant codec. For example, the same motherboard can be used to provide two different audio SKUs, one with basic stereo support and another with support for 7.1 surround sound. The different capabilities result from using a different codec for each SKU.

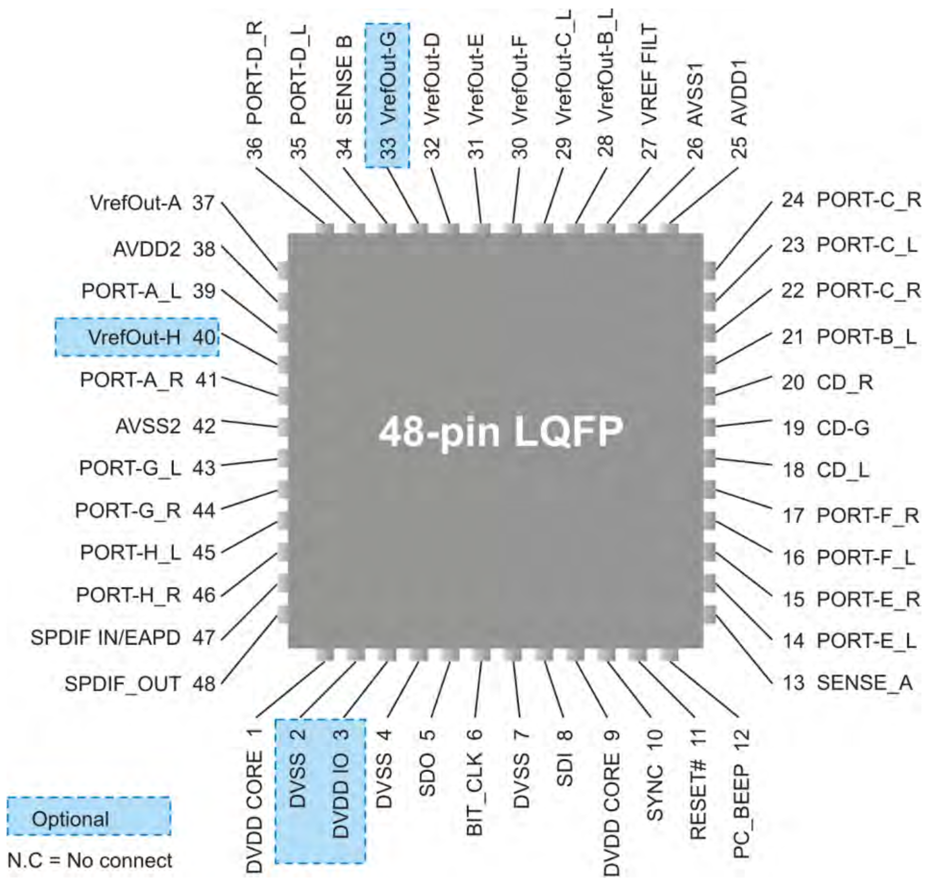


Figure 4.16 Recommended Pinout for 48-Pin Codec Package

Pins, Pin Widgets, and Ports

The word *pin* can be used in different ways when referring to Intel HD Audio systems. Audio software engineers use "pin" in many different contexts, some having nothing to do with audio. For purposes of this discussion, the term applies in the hardware context. But remember that if you hear a software guy using the word pin, it might mean something other than what is described here.

For integrated circuits, the word "pin" has traditionally been used to refer to a physical pin on a codec that is soldered down to a pad on the motherboard. The term *Pin Complex* was introduced in the Intel HD

Audio specification to refer to a group of four or more hardware pins which are functionally linked together. You have an analog signal pin for the left stereo channel, an analog signal pin for the right stereo channel, a pin for a switchable microphone bias source, and a SENSE pin that is shared between groups of four analog ports. This functionality context is shown in Figure 4.17. These pins on the codec are often not located together physically, and they may be spread around all four sides of the codec package. See Figure 4.16 for more detail on individual pin locations.

"Pin widget" and "pin complex widget" both refer to the widget control node which controls a particular pin complex. The word *Port* is used to describe the group of physical pins that is associated with an input or output pin widget complex on the codec.

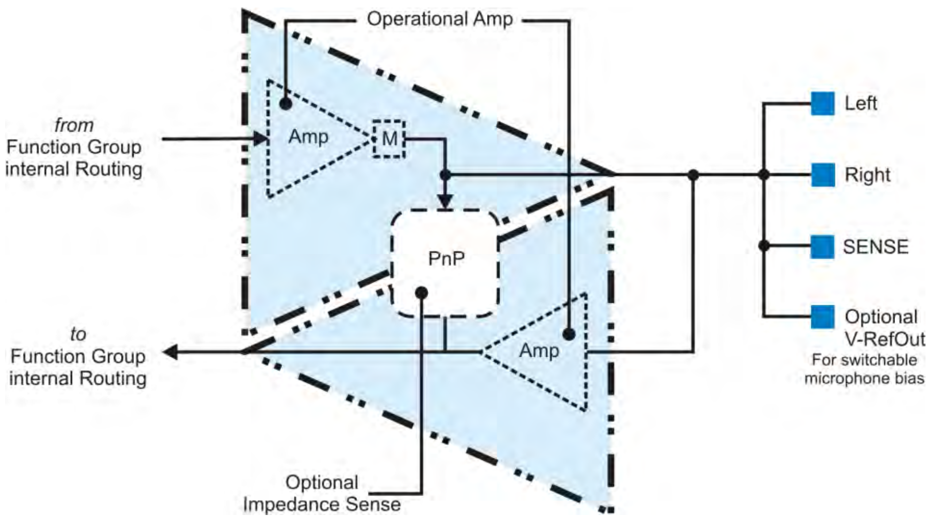


Figure 4.17 A Retaskable Pin Widget with its Related Port Shown on Right

For example, in Table 4.5 you can see that the physical pin numbers for Port A are 39, 41, 37, and 13. Pin 39 is the left audio channel, and pin 41 is the right audio channel. Pin 37 is the switchable microphone bias source (VRefOut_A), and pin 13 is the SENSE pin that is shared with Ports B, C, and D. Microphone bias is not available on ports G and H for this particular codec. Ports A through D share one SENSE pin, while ports E through H share a separate SENSE pin.

The designers of the codec in Figure 4.12 chose to use Widget IDs with hexadecimal values that match the alphabetical port names for the first 6 ports. While this choice is a nice touch that makes it a bit easier to work with ports A thru F, you should not expect to see this type of node-numbering on all codecs.

Table 4.6 combines information from the widget diagram shown in Figure 4.12, the codec pinout shown in Figure 4.16, and Intel and Microsoft desktop platform recommendations for port assignments and functional color codes. The system should include an Intel HD Audio codec with at least eight analog ports, at least five stereo DAC pairs, and at least two stereo ADC pairs. This configuration is recommended for desktop PCs that are currently in design and production. This example is fully compliant with the Windows 7 Logo program, including multi-streaming support for Real Time Communication (RTC) applications such as VoIP or instant messaging.

You can make your life a lot easier by making a version of this table for each system you are working on, and filling in the appropriate Node ID numbers, hardware pin numbers, colors, and port usages. Check widget node IDs, schematic drawings, jack location and color, and pin configuration settings against this chart. Doing so provides a solid foundation for the pin configurations described next.

Table 4.6 **Example of Ports, Pin Widget Node IDs, Pins, Colors, and Usages**

Port	Pin Widget Node ID	Codec HW Pins	Color	Recommended Usage
A	0xA	39,41,37,13	Green	Front Panel Headphone Output
B	0xB	21,22,28,13	Pink	Front Panel Microphone Input
C	0xC	23,24,29,13	Blue	Rear Panel Line Input
D	0xD	35,36,32,13	Green	Rear Panel Front L/R pair (for 7.1)
E	0xE	14,15,31,34	Pink	Rear Panel Microphone Input
F	0xF	16,17,30,34	Black	Rear Panel Back L/R pair (for 7.1)
G	0x10	43,44,34	Orange	Rear Panel Center/LFE pair (for 7.1)
H	0x11	45,46,34	Grey	Rear Panel Side L/R pair (for 7.1)
S/PDIF Out	0x21	48	Black	Digital Out
S/PDIF In	0x22	47	Black	Digital In
CD In	0x12	18,19,20	Black	CD In (ATAPI connector on M/B)

The Pin Widget Node ID is usually different for each model of Intel HD Audio codec. This particular table matches the widget diagram in Figure 4.12, and the pinout shown in Figure 4.16. Items highlighted in grey are optional.

Verb Tables

A verb table is a list of 32-bit Intel HD Audio codec command sequences that are compiled into the BIOS as data. Each verb or command to be executed is hard-coded in assembly language as a list of 32-bit data values. This table allows the BIOS to transmit a variable-size list of codec commands in what appears to be a single operation. No mechanism for querying the codec or for error checking exists. The verbs are transmitted blindly. If a particular verb or a particular Node ID is not supported, the codec should skip over those verbs without complaint.

While not a part of the Intel HD Audio specification, verb tables are a very practical part of Intel HD Audio system implementation, and they are supported by most major BIOS vendors that support Intel HD Audio. Verb tables are defined in the *ICH7 Intel HD Audio Programmer's Reference Manual* (PRM); you can find a link to it on this book's companion Web site.[1] Verb tables are usually delivered as text files or code files, and are copied and pasted into the BIOS source code.

The verb table allows the audio team to provide the BIOS engineer with a table of codec commands to execute during system startup and during a return from S3 sleep state, while isolating the BIOS engineer from the details of Intel HD Audio codec implementations and motherboard schematic design. Completion of the verb table integration into the BIOS is a major milestone in the audio system development, since it should allow the Microsoft UAA class driver for Intel HD Audio to be used for bringing up the hardware. The "UAA Class Drivers" section in Chapter 7 contains additional details on the Microsoft UAA class driver for Intel HD Audio.

While verb tables are a handy way to deliver this information to the BIOS developer, verb tables are not particularly readable by humans. For this reason, it is important to provide as many comments as possible along with the code in the verb tables. In the absence of comments, deconstructing verb tables is a lengthy chore requiring your colleagues to consult the Intel HD Audio specification, the codec data sheet, the system schematics, and hexadecimal and binary calculators. This task is not trivial! Provide comments often and in as much detail as possible. The examples in this section contain an appropriate level of detail.

[1] The verb table and code examples in the currently available PRM describe an earlier version of the verb tables that actually are incorporated in many currently available BIOS distributions. Some of the details on how the header is formed are now out of date, but the way that the verbs are defined and placed in the tables remains current.

If the motherboard design has stuffing options for more than one codec, the BIOS for that motherboard should contain one verb table for each of the different codecs that might be installed, each of which contains the codec ID as the first entry in the table. The BIOS reads the codec ID and matches it to a BIOS table in this fashion.

To be compliant with the Microsoft UAA class driver for Intel HD Audio under Windows 7, the BIOS must include a verb table which contains default pin configuration settings for *all* pin widgets in the codec, regardless of whether they are connected to anything or not. The BIOS must also program the subsystem ID register in the Audio Function Group (AFG). Since each of these registers is a 32-bit register, four verbs are used to program each register, byte by byte. In the case of a codec with eight pin widgets, a minimum of 36 verbs are necessary to program the codec at system startup.

Figure 4.18 shows the first set of four verbs in the verb table, which programs the subsystem ID register. This example sets the 32-bit Subsystem ID register to an arbitrary value of `0xAABBCCDD`. (See Chapter 7 for info on what values to use in a real system.) The four different verbs in this list are all directed to Node ID `0x01` on Codec ID#2, which is connected to the SDI#2 pin of the Intel HD Audio controller. This node is the audio function group for the entire codec, where the Subsystem ID register is located.

```
;Audio function group  (NID=01h), SDI#2
; set the 32-bit subsystem ID by writing four bytes to
; successive addresses. This examples writes 0xAABBCCDD
; to the subsystem ID register of the codec
;
: the four verbs to accomplish this are defined below
; with each of the 32-bit verbs stored as 32-bit data

      dd    201720DDh        ; Set bits  7:0  to 0xDD
      dd    201721CCh        ; Set bits 15:8  to 0xCC
      dd    201722BBh        ; Set bits 23:16 to 0xBB
      dd    201723AAh        ; Set bits 31:24 to 0xAA
```

Figure 4.18 Simple Verb Table Written in Assembly Language

To understand this list better, try breaking the first verb into its component parts:

```
2 01 720 DD
       0x2   = Codec ID number, that is the number of the
```

```
        SDI Pin to which the codec is wired
0x01  = Node ID number
0x720 = 12-bit command: write to bits 0:7 of
        the subsystem ID
0xDD  = 8-bit payload
```

The second command in Figure 4.18 works in the same way, but uses a verb code of `0x721` to write the payload of `0xCC` to bits 15:8 of the subsystem ID register in the codec, and so on down the list. Interestingly enough, it is possible to read all 32 bits of most codec registers in a single operation, but writing to them usually requires four separate operations.

Sending Verb Tables to the Codec

At startup, the BIOS reads the codec vendor ID and the codec ID, and possibly the codec revision ID, and then searches through a list of verb tables until it finds a verb table with a matching codec ID. If successful, the BIOS transmits the entire verb table to the codec. It also does this on returning from the S3 state.

A BIOS intended for desktop motherboards may also include logic or multiple verb tables to determine how to program pin widgets allocated to the front panel connectors that are optional on many new models. The BIOS can use a GPIO on the Southbridge to determine whether a front panel dongle is present. (See more details in Chapter 5, "Front Panel Considerations.") Additional logic in the BIOS is used to determine how to configure the pin widgets servicing the front panel. If the system has no front panel connectors, the BIOS sets the pin widgets to No Connectivity. If the front panel dongle is the older AC97 style dongle, with no presence to detect, the pin widgets are programmed to indicate no jack detect capability. Be aware that this implementation is not Windows 7-compliant.

While the driver normally is charged with maintaining the codec state during S3, you have no guarantee that a driver will be present and loaded during the transition into and out of S3, so for consistency the BIOS must always retransmit the verb table any time that power to the codec is removed and then restored. Otherwise, the Plug and Play ID becomes corrupted after resuming from S3, and the driver is unable to load.

If the system is using the codec hardware defaults or if the codec's digital power supply is kept powered up during S3, re-transmitting the verb tables when coming out of S3 is unnecessary; in all other cases the BIOS should retransmit the verb tables.

When coming out of S3, you have an additional consideration if a modem codec or other Intel HD Audio device generates a wake event. Because the OS needs to know which device caused the wake event, the BIOS must perform the following sequence when resuming from S3.

1. Store the contents of the controller's STATESTS State Change Status register, which contains info on which (if any) Intel HD Audio device caused the wake event.
2. Take the controller out of reset, program the configuration default and optionally, program the mute registers.
3. Restore the original STATESTS register contents.
4. Put the controller and Intel HD Audio link back into reset.

This sequence should restore everything in the codec to the state it was in when the wakeup event occurred, with the exception of the new values contained in the verb table.

Muting and Startup Work-arounds

One of the requirements for Windows 7 Logo is for the BIOS to mute outputs during the power-up sequence. While not specifically defined in the PRM, the verb table mechanism can be used for this purpose. To mute a widget, you must send a verb which specifies the codec ID, the widget node ID, a Set Amplifier Gain/Mute command (`0x3`) and a 16-bit payload of `0xF080`. For instance, to mute the widget at Node ID `0x04` on codec ID #2, you would send `0x2043F080`. This value mutes both input and output volumes on both the right and left channels of the widget, as well as setting the volume controls on each pin widget to their lowest settings.

It's important to send these mute messages to the proper widgets. By studying Figure 4.12, you can see that the volume controls are implemented in the DAC widgets rather than in the pin widgets, which means that in this codec a mute command sent to a pin widget is simply ignored. Instead, you must send series of mute commands to all the DACs in the codec, as shown in Figure 4.19.

```
; MUTE ALL DACS on Codec ID #2
;
; the verbs are formed in groups of four in case the BIOS
; uses the count of pin widgets to determine how big the
; verb table is. The pin widget count should be increased
; by two to accommodate these additional commands
; The second group of 4 includes some repeated to make 4
;
dd    2023F080h         ; Mute DAC at Node 0x02
dd    2033F080h         ; Mute DAC at Node 0x03
dd    2043F080h         ; Mute DAC at Node 0x04
dd    2053F080h         ; Mute DAC at Node 0x05

dd    2063F080h         ; Mute DAC at Node 0x06
dd    2063F080h         ; Mute DAC at Node 0x06 (repeat)
dd    2063F080h         ; Mute DAC at Node 0x06 (repeat)
dd    2063F080h         ; Mute DAC at Node 0x06 (repeat)
```

Figure 4.19 List of Eight Verbs Which Will Mute All DACs in the Codec

Unlike the pin-configuration registers, the mute for each pin widget can be set with a single 32-bit verb, rather than a set of 4 verbs. If the BIOS requires you to specify the number of pin widgets in the verb table, you can fool it into supporting mute verbs by telling it that you have one or two additional pin widgets. You would add one to this count for every four mute verbs that you want to send. If you have an odd number of mute verbs to send, simply repeat the last one until they line up in sets of four. Some BIOS versions may use a terminator such as 0xFFFFFFFF to signify the end of the verb table, rather than specifying a count in the header. In this case, you can easily add as many verbs as you like without needing to restrict them to sets of four verbs.

Be sure to account for analog PC beep if it is implemented. For instance, if you want to be able to hear POST tones coming from the stereo outputs on the rear of the system, don't send a mute verb to that pin widget. Instead, you might send a verb that configures the output appropriately for the intended usage. If you are purposefully un-muting an output so that POST tones can be heard at startup, DO NOT set the volume control to unity gain. Instead, set it to 25 or 30 decibels of attenuation, so that if the system is attached to a high-powered speaker system, that the POST tones don't damage the speakers or the listener's ears. See "Hardware Volume Scaling" earlier in this chapter for more details on how to do so.

While the pin widgets may not contain volume controls, you can approximate a mute in many cases by disabling the pin widget's input and output circuits by writing to the Pin Widget Control register for each pin widget, as shown in Figure 4.20.

```
; Set all Pin Widgets on Codec ID #2
; to disable input, output, and VRefOut
;
; the verbs are formed in groups of four in case the BIOS
; uses the count of pin widgets to determine how big the
; verb table is. The pin widget count in the BIOS should
; increase by two to accommodate these additional verbs
;
dd    20A70700h         ; Disable Pin Widget 0x0A
dd    20B70700h         ; Disable Pin Widget 0x0B
dd    20C70700h         ; Disable Pin Widget 0x0C
dd    20D70700h         ; Disable Pin Widget 0x0D

dd    20E70700h         ; Disable Pin Widget 0x0E
dd    20F70700h         ; Disable Pin Widget 0x0F
dd    21070700h         ; Disable Pin Widget 0x10
dd    21170700h         ; Disable Pin Widget 0x11
```

Figure 4.20 Verb List to Disable Input, Output and VRefOut for All Pin Widgets

Notice that disabling a pin widget or muting the codec is not guaranteed to be a noise-free event, although it should be the case for a well-designed codec. In some cases, especially with a poorly designed codec, you might find that setting the mutes or disabling the pin widget ends up causing more noise than if you didn't write to the widget all. A number of factors come in to play in this situation, including the power supply sequencing and the rise time of the codec's DC levels for analog output and VRefOut.

Also remember that when the driver loads it may reset the Intel HD Audio link. If this happens, the codec registers return to their default state. This reset could also cause a pop or noise when abruptly changing from the settings written by the verb table at startup. You can isolate each noise source by disabling the audio driver, then removing power from the entire system.

After waiting at least one minute for all capacitors to drain off, start the system while listening through a good pair of speakers or headphones. Any noise that you hear is a function of the codec coming out of reset, the power supplies starting up, or the verb tables being written to the codec. Try changing the verb tables to see if this changes the noise behavior. Once you've characterized the startup noise sources, re-enable the driver. Any noises that you hear when the driver is started (and the codec is reset) could have an interaction with the verb table s programmed at startup. Try leaving out messages to the Amplifier/Mute register and the Pin Widget Control to further identify any interaction.

This ability to arbitrarily configure the codec at system startup can be very useful in cases where you are having problems with pops, clicks, or noises at startup before the driver loads. If you are using the Microsoft UAA class driver for Intel HD Audio, this period before the driver loads may be one of the few places where you can address these pops and clicks. You may want to experiment with other settings of the various pin widget controls. Each make and model of codec is likely to have some variation in how it responds to various settings while being powered up and initialized.

Sometimes, special cases require other verbs to be sent to the codec at initialization time, before the driver has been loaded, to address specific behaviors. Just about any command to the codec can be routed through verb tables, as long as it doesn't require any knowledge of the current codec state.

Pin Configuration Registers

The Pin Widget or Pin Widget Complex mentioned earlier has a special place in the relationship between Intel HD Audio and Microsoft's Universal Audio Architecture (UAA). The Microsoft UAA class driver for Intel HD Audio and some codec-vendor-specific drivers use the information that the BIOS programs into the Pin Configuration registers to determine the schematic design and layout of the audio subsystem and to automatically configure the driver topology based on the pin-configuration registers. Future versions of Linux drivers might also make use of these registers.

To meet Windows 7 Logo requirements, the BIOS must program valid pin-configuration defaults into each pin widget contained in the codec. The codec hardware also has default configurations hard-coded into each of the codec's pin widgets, but these default configurations are only valid if the motherboard has been wired identically to the schematic used to generate the configuration defaults.

In almost all cases, the motherboard design does not match the codec's default pin configurations, so the BIOS has to program the pin-configuration registers during system boot-up. Often, the codec vendor provides the pin configuration defaults as part of the schematic review process. Like the Subsystem ID, each 32-bit configuration register requires four separate 8-bit writes to set the register, as shown in Table 4.7.

Table 4.7 Pin Configuration Default Register Fields

31:30	29:24	23:20	19:16	15:12	11:8	7:4	3:0
Connectivity	Location	Device	Connector	Color	Misc	Association	Sequence

The grey shading indicates each of the 8-bit segments used when writing to this register

The BIOS must therefore send four verbs for the subsystem ID, and an additional four verbs for each pin widget or port on the codec, regardless of whether the port is connected to anything or not. Figure 4.21 shows an example of a partial verb table, which sets the pin configuration defaults that are shown in Table 4.8. A similar sequence must be repeated for each pin widget in the codec, whether it is connected to anything or not.

Be aware that the Subsystem ID Register and the Pin Configuration Default registers in each pin widget are not touched during a reset from any source, while other registers in the codec are reset to their default values upon a function group reset. The contents of these registers are only lost when digital power is removed from the codec; the BIOS must restore these registers when power is restored to the codec's digital power supply.

```
; Front Panel HP Out uses port A (NodeID = 0xA, SDI#2)
; Set Node ID 0x0A Pin Configs to 0x02214070
dd 20A71C70h ; Set 7:0   to 0x70 - Assoc 7, Seq 0
dd 20A71D40h ; Set 15:8  to 0x40 - Green, Jack Detect
dd 20A71E21h ; Set 23:16 to 0x21 - HP Out, 3.5 mm jack
dd 20A71F02h ; Set 31:24 to 0x02 - Front, chassis
```

Figure 4.21 Verb table and Pin Configuration for an Independent Stereo HP Out Jack on Front Panel

Table 4.8 Pin Configuration for an Independent Headphone Out Jack on Front Panel

Connectivity	Loc	Device	Type	Color	Jack Detect	Assoc	Seq
Chassis Jack	Front	HP Out	3.5 mm	Green	Yes (0)	7	0

For the most part, the individual fields of the Configuration Default register are independent of each other. These fields are sufficient to describe a static configuration, but they do not contain enough information to fully describe a retaskable jack.

The Port Connectivity field, shown in Table 4.9, allows you to specify whether the port is disconnected, connected to a jack, or connected directly to an integrated device such as a microphone or speaker amplifier. Though the Intel HD Audio specification allows for both a jack and an internal device to be attached to a single port, this configuration is not fully supported by the Microsoft UAA class driver for Intel HD Audio and should be avoided.

Table 4.9 Port Connectivity (Bits 31:30) of Pin Configuration Default Register

Bits 31:30	Value
00b	Port is connected to a jack (3.5 mm, ATAPI, RCA, etc)
01b	No physical connection for Port
10b	A fixed function device (integrated speaker, integrated mic, etc.) is attached.
11b	Both a jack and an internal device are attached. (Not recommended)

Note: Port connectivity settings that are highlighted in gray are not recommended.

The Location field consists of two subfields. Bits 29:28 are used to indicated whether the port is on the primary chassis, an external chassis, internal to the PC (not user accessible), or “Other,” which includes microphones on the PC lid. Bits 27:24 are interpreted differently based on the contents of bits 29:24. This alternative results in a very large table of supported devices when decoded, but only a small subset of these possible devices is actually used in modern PCs. Most motherboard designs use only those locations not highlighted in gray in Table 4.10.

The Microsoft UAA class driver for Intel HD Audio uses the locations to build logical names for jacks, as well as to group related topologies. WHQL tests enforce Logo requirements by ensuring that two or more jacks of the same color are never in the same location or in the same association. It’s OK to have a pink microphone input jack on the front, and another pink microphone input jack on the rear, but it’s not OK to have two pink jacks on the front, next to each other.

When the Port connectivity bits are set to “jack” (`0x00`), the value in the high byte of the Pin Configuration Default register directly reads out the location. This result is true for any byte value for bits 31:24 that comes up to less than `0x40`.

Table 4.10 **Location / Connectivity (Bits 29:24) of Configuration Default Register**

b31:24	b31:30	b29:28	b27:24	Value
01h	00b	00b	1h	External jack on primary chassis: Rear
02h	00b	00b	2h	External jack on primary chassis: Front
03h	00b	00b	3h	External jack on primary chassis: Left
04h	00b	00b	4h	External jack on primary chassis: Right
05h	00b	00b	5h	External jack on primary chassis: Top
06h	00b	00b	6h	External jack n primary chassis: Bottom
08h	00b	00b	8h	External jack on primary chassis: Drive bay
18h	10b	01b	8h	HDMI Integrated (Direct S/PDIF trace to HDMI Encoder)
19h	00b	01b	9h	ATAPI Jack on motherboard
90h	10b	01b	0h	Internal: Speaker
B7h	10b	11b	7h	Other: Mic Inside Mobile Lid
B8h	10b	11b	8h	Other: Mic Outside Mobile Lid

Note: Uncommon combinations are highlighted in gray, and should be avoided. See Intel HD Audio Specification for complete description of the Location field.

The Default Device field shown in Table 4.11 describes the different functions that might be connected to a port. You rarely use any of the devices highlighted in gray in a modern motherboard. Items highlighted in gray are not supported by the Microsoft UAA class driver for Intel HD Audio.

Be sure to use speaker category only for devices that drive an amplifier that is outside the codec but inside the PC chassis. Either the speaker should be in the PC chassis as well, or it should be a passive 8-ohm speaker connected directly to speaker clips on the PC chassis, much like a stereo receiver. Devices such as self-powered 5.1 speaker systems that are connected through line-out jacks should be designated as line-out, not as speaker.

Table 4.11 **Default Device (Bits 23:20) of Pin Configuration Default Register**

Bits 23:20	Value
0h	Line Output Jack
1h	Internal Speaker (amplifier is built into PC chassis)

Bits 23:20	Value
2h	Headphone Output Jack
3h	*Analog CD Input (via ATAPI connector)*
4h	S/PDIF Out (optical or coaxial)
5h	Digital Other Out
6h	Modem Line Side
7h	Modem Handset Side
8h	Line Input Jack
9h	*Aux*
Ah	Microphone Input
Bh	Telephony
Ch	*S/PDIF In* (optical or coaxial)
Dh	Digital Other In
Eh	Reserved
Fh	Other

Note: Items highlighted in gray are not supported by the UAA class driver. Less common devices shown in italics should be avoided when possible.

The Connection Type field detailed in Table 4.12 describes the type of connector that is used. In practice, the highlighted subset of these connector types is rarely used on modern PCs.

The Microsoft UAA class driver for Intel HD Audio sets the outputs to a fixed maximum volume output level when an RCA jack is specified and the device type is set to Line Out. It labels the audio endpoint as *line connectors*, to indicate a multi-channel A/V receiver.

An RCA jack can also be used as digital connector for S/PDIF signals. The same connector code (`0x4`) is used in this case, but the device field is set to either S/PDIF In or S/PDIF Out, rather than Line In or Line Out. The Microsoft Intel HD Audio UAA Class Driver handles volume controls differently when an RCA jack is used, and it also displays a different name for the audio endpoint, which is typically a multi-channel A/V receiver.

The XLR connector is very useful for connecting professional microphones. It is typically mounted in a 5-inch wide content-creation audio bay on the front of the PC, in the spot where a CD or hard drive might otherwise be mounted. The XLR connector does not inherently have a jack detection mechanism, so this type of input should be treated as always connected.

Even though it's not indicated in the table, it's OK to specify a quarter-inch headphone jack on a content creation bay. As long as the quarter-inch jack is wired the same way that the 3.5-millimeter jack is wired for jack detection, this configuration will work well. Other standard quarter-inch jack configurations used in the audio industry do not match the way that 3.5-millimeter jacks are used on the PC. For instance, no pro-audio equipment uses a stereo quarter-inch phone jack for stereo line in. Instead, two quarter-inch jacks would be used for that purpose. In the audio industry, quarter-inch stereo jacks are more likely to be connected in a balanced configuration, which is rarely used in PC designs. Using quarter-inch jacks improperly can cause lots of confusion, and should be avoided.

Future versions of the Microsoft UAA class driver for Intel HD Audio could make use of these connector types, but these settings have little effect on the UAA class driver that ships with Windows 7.

Table 4.12 **Connection Type (Bits 19:16) of Pin Configuration Default Register**

Bits 19:16	Value
0h	Unknown
1h	3.5 mm stereo/mono phone jack (incorrectly referred to as 1/8”)
2h	¼” stereo/mono phone jack
3h	ATAPI Internal
4h	RCA jack (may be used for analog audio or for coaxial S/PDIF)
5h	EIAJ Optical or TOSLINK† connector for S/PDIF or ADAT
6h	Other Digital
7h	Other Analog
8h	Multi-channel Analog (DIN)
9h	XLR/Professional
Ah	RJ-11 (Modem)
Bh	Combination
Ch - Eh	Undefined
Fh	Other

Note: Uncommon connectors are highlighted in grey, and should be avoided.

Some classes of Microsoft 7 Logo compliance call for specific colors to be used for the various standard analog audio inputs and outputs on a PC. In addition to defining these colors, the Intel HD Audio specification also defines some additional colors which are not used for audio jacks on systems conforming to Windows 7 Logo requirements. Avoid colors that are highlighted in Table 4.13 to help prevent WHQL failures.

Table 4.13 **Color (Bits 15:12) of Pin Configuration Default Register**

Bits 19:16	Value	Standard Usage	Associated Device
0h	Unknown	n/a	Don't Use
1h	Black	Rear Surround L/R	Line Out
2h	Grey	Side Surround L/R	Line Out
3h	Blue	Line In L/R	Line In
4h	Green	Line Out or HP Out L/R	Line Out or HP Out
5h	Red	Not used for audio 3.5mm jacks	Don't Use
6h	Orange	Center/LFE	Line Out
7h	Yellow	Not used for audio 3.5mm jacks	Don't Use
8h	Purple	Not used for audio 3.5mm jacks	Don't Use
9h	Pink	Microphone in mono or stereo	Microphone In
Ah - Dh	Reserved	n/a	Don't Use
Eh	White	Not used for audio 3.5mm jacks	Don't Use
Fh	Other	n/a	Don't Use

Note: Colors that are not Windows 7 Logo-compliant are highlighted in gray. Avoid using any of these colors in your pin configuration defaults.

Three of the four bits designated as Misc are currently undefined, and only one of the four is defined. This Jack Detect override bit is tricky, because it uses negative logic.

If the Jack Detect override bit is set, it indicates to the audio driver that it is not possible to detect jack insertion events on this jack. This result could be from using a connector that does not include an isolated switch, such as an RCA or XLR connector.

To meet Windows 7 Logo requirements, all 3.5-millimeter analog jacks must set the Jack Detection Override bit designated in Table 4.14 to zero, indicating that Jack Detection is present both in the codec and that the jacks on the motherboard have switches and are wired properly, even for front panel jacks. Analog outputs using RCA jacks and a device type of Line Out may set this bit to one, indicating that a switch on the jack is not present.

Table 4.14 Misc (Bits 11:8) of Pin Configuration Default Register

Bits 11:8	Value
00h	Jack detection through SENSE pins
01h	No jack detection implemented
2h – 7h	Reserved

Note: The 1.0 Intel HD Audio specification defines only bit 8 of the Misc fields. Bit 8 should be set whenever jack detection is not implemented for any reason. Setting Bit 8 on analog I/O ports may result in a WHQL failure under Windows 7

Associations and Sequences

Each audio endpoint in complete system is defined by a separate association and sequence. Associations in the range of `0x1` through `0xE` can be used for multi-channel or stereo applications. Association `0xF` is reserved only for devices which expose a single pin widget. Each association must have a unique number. Although several pin widgets might use Association `0xF`, the Microsoft UAA class driver for Intel HD Audio treats each of the pin widgets in this association as a separate stereo device. The associations are evaluated in priority order, with association `0x1` having the highest priority and association `0xF` having the lowest priority.

The association is stored in Bits 7:4 of the Pin Configuration Default register. Associations that describe multi-channel audio streams can be represented by a 4-bit hexadecimal number from `0x01` to `0x0E`. Association `0x00` is reserved, and association `0x0F` is limited to a single pin widget. The sequence is stored in Bits 3:0 of the Pin Configuration Default register. A sequence must be unique in any particular association. Some special sequence values also are used to indicate specific behavior.

Associations built from a single pin widget should always use #0. Examples are Stereo Headphone Out, Stereo Line Input, Stereo (or mono) microphone input, Stereo Line Out, Stereo S/PDIF In, and Stereo S/PDIF Out. Multi-channel streams can be created by setting several pin widgets to the same association, and then using the Sequence numbers to define the individual channel pairs in the stream.

Associations can also be used to describe ADC multiplexer (Mux) inputs, ADC mixer (mix) inputs, mic arrays, and redirected headphone configurations. The audio driver uses the associations to determine how to make connections and allocate resources inside the codec. The audio driver evaluates each Association in numerical priority, starting with Association 0x01 and ending with Association `0x0F`.

Table 4.15 shows a possible set of association assignments for a complete system. If you prefer, you can mix and match any way that you would like, but you should try to set up a numbering scheme that is consistent across multiple systems, for ease of understanding, and that takes association priorities into account. Starting with one and skipping every other association number, as shown in the table, allows you to easily reorder priorities without renumbering the entire table.

Table 4.15 **Example of a Set of Association Assignments in a Typical System**

Association	Value
0h	Reserved, Do Not Use
1h	Integrated Stereo Speaker Pair
3h	Rear Panel Primary Line Out (may be stereo, 5.1, or 7.1)
5h	Rear Panel Primary Stereo Line In or ADC Mux In
7h	Front Panel Secondary Stereo HP Out
9h	Front Panel Secondary Stereo Mic In Jack
Ah	Rear Panel Stereo S/PDIF Out
Ch	Rear Panel Stereo S/PDIF In

Note: No standard requires this ordering, but you may wish to keep consistent between different models for ease of understanding.

One or more examples of each of these assignments follow. The association numbers shown in the examples match Table 4.15. You can use these examples to create your own verb table s directly.

Stereo Stream Associations

An association consisting of a single pin widget should always specify a sequence of zero. The association number should be unique, with the exception of `0x0F`, which supports multiple stereo associations. The example in Table 4.16 shows an association consisting of a single blue 3.5-millimeter stereo Line In jack on the rear panel of the computer.

Table 4.16 **Pin Configuration for Rear Panel Line Input**

Connectivity	Loc	Device	Type	Color	Jack Detect	Assoc	Seq
Chassis Jack	Rear	Line In	3.5 mm	Blue	Yes (0x0)	5	0

Note: See Appendix A for complete verb tables for this association.

While Table 4.16 shows the entire contents of the pin widget's default configuration register, this information is not enough by itself. The entries in the actual verb table also need to know the codec ID and the Node ID of the pin widget. For your convenience, completely assembled verb tables for each of the Pin Configurations shown in this chapter are available in Appendix A. These examples all assume that the Codec ID is #2 and that the Node IDs of the codec are the same as those shown in the widget diagram in Figure 4.12, where you can see that Node ID `0x0C` is used for Port C.

In this example, the pin widget attached to Port C is targeted. The Codec ID of `0x2` and the Node ID of `0x0C` can be seen at the beginning of each of the four commands in the verb list in Figure 4.22.

```
; Rear Panel Line In is port C (NodeID = 0x0C, SDI#2)
; Set Node ID 0x0C Pin Configs to 0x01813050
dd 20C71C50h ; Set 7:0   to 0x50 - Assoc 5, Seq 0
dd 20C71D30h ; Set 15:8  to 0x30 - Blue, Jack Detect
dd 20C71E81h ; Set 23:16 to 0x81 - Line In, 3.5 mm jack
dd 20C71F01h ; Set 31:24 to 0x01 - Rear, chassis
```

This verb list matches the line inputs in Table 4.16.

Figure 4.22 Verb List for Rear Panel Line Input Configuration

Table 4.17 shows the pin configuration for a green line-out jack on the rear of the computer. In this case port D is used, which is also assigned to Node ID 0x0D in the codec. In the corresponding verb table in Appendix A, you can see the "20D" at the beginning of each verb in the table. This pin configuration assumes that jack detection is wired correctly on the motherboard.

Table 4.17 **Pin Configuration for Rear Panel Line Output**

Connectivity	Loc	Device	Type	Color	Jack Detect	Assoc	Seq
Chassis Jack	Rear	Line Out	3.5 mm	Green	Yes (0x0)	3	0

Note: See Appendix A for an example of a completed verb table for this association.

Table 4.8 showed the pin configuration for a green headphone jack on the front of the computer. Node ID `0x0A` is used for this pin widget. Notice that a differently numbered association is used to denote the front panel headphone jack from the rear line out. This differentiation provides the option for a multi-streaming output configuration. This pin configuration assumes that the specified port is capable of driving headphones, that coupling capacitors large enough to drive headphones are present, and that jack detection has been wired correctly on the motherboard.

Table 4.18 shows the pin configuration for an independent microphone input jack using Node ID `0x0B`, which in this case is also Port B. This configuration assumes that the microphone bias circuit and the jack detection circuit are configured properly on the motherboard.

Table 4.18 **Pin Configuration for Front Panel Microphone Input**

Connectivity	Loc	Device	Type	Color	Jack Detect	Assoc	Seq
Chassis Jack	Front	Mic In	3.5 mm	Pink	Yes	9	0

Note: See Appendix A for an example of a completed verb table for this association.

Table 4.19 describes a digital S/PDIF optical output jack using Node ID 0x15. No SENSE pin is associated with S/PDIF Out, so the Jack Detect bit is set to `0x1`, which indicates that jack detection is not available for this jack.

Table 4.19 **Pin Configuration for Rear Panel S/PDIF Output**

Connectivity	Loc	Device	Type	Color	Jack Detect	Assoc	Seq
Chassis Jack	Rear	S/PDIF Out	Optical	Black	No (0x1)	A	0

Note: See Appendix A for an example of a completed verb table for this association.

You should be able to put together a complete system featuring separate stereo input and output circuits on front and rear panels by using the preceding examples. Be sure to check the codec data sheet for the proper Node IDs for each pin widget, as they vary from codec to codec.

5.1 Surround Multi-channel Stream Association

Surround sound output with five satellite channels and a subwoofer can be accomplished by assigning three stereo line-out ports to the same association, but with unique sequence numbers for each pin widget in the association. Sequence numbers often have unique meanings; you cannot assign them arbitrarily as you can with association numbers. Additional information on multi-channel speaker configurations can be found in Chapter 7.

When used together in a single association, the set of sequence numbers (0, 1, 2) defines a 6-speaker configuration as shown on the left side of Figure 4.23, with speakers located at the Front Left (FL), Front Right (FR), Front Center (FC), Low Frequency Effects (LFE), Back Left (BL), and Back Right (BR) positions.

Similarly, sequence numbers (0, 1, 4) represent a 6-speaker configuration with speakers located at the FL, FR, FC, LFE, Side Left (SL), and Side Left (SR) positions, as shown on the right side of Figure 4.23.

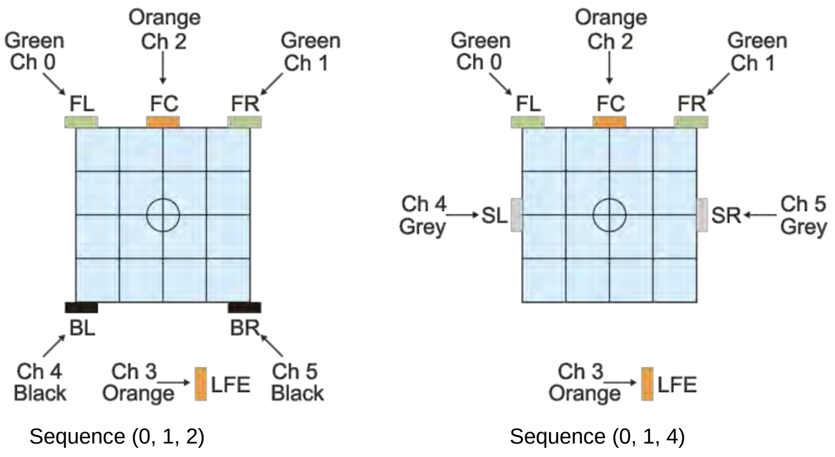


Figure 4.23 5.1 Surround Sequences for a Multi-channel Association

The Speaker Configuration control panel in Windows does not distinguish between the two 5.1 speaker configurations shown in Figure 4.23, which differ only in whether BL and BR speakers are used in place of SL and SR speakers. The control panel uses the label *5.1 surround sound speakers* to identify either configuration. The reason for not distinguishing between the back-speaker and side-speaker configurations in the control panel is that most listeners don't distinguish between these speaker positions, so having two different settings doesn't make a lot of sense.

Sequence (0, 1, 4) is the preferred configuration for Windows 7. This configuration assumes that all ports in the association have jack detection circuitry implemented on the motherboard. Table 4.20 details the settings for this configuration.

Table 4.20 **Pin Configuration for 5.1 Surround Line Outputs Using Sequence (0, 1, 2)**

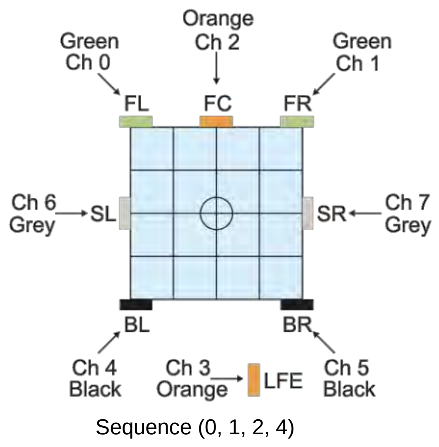
Connectivity	Loc	Device	Type	Color	Jack Detect	Assoc	Seq
Chassis Jack	Rear	Line Out	3.5 mm	Green	Yes (0x0)	3	0
Chassis Jack	Rear	Line Out	3.5 mm	Orange	Yes (0x0)	3	1
Chassis Jack	Rear	Line Out	3.5 mm	Black	Yes (0x0)	3	2

Note: The choice of sequence number determines the function of each port, which must be matched with the proper color for that function. See Appendix A for an example of a completed verb table for this association.

7.1 Surround Multi-channel Stream Association

Sequence numbers (0, 1, 2, 4) represent an 8-speaker configuration with speakers located at the FL, FR, FC, LFE, BL, BR, SL, and SR positions, as shown in Figure 4.24. Sequence 3 must not be used in this configuration. The *7.1 home theater speakers* configuration should be selected in the Advanced Audio Properties Speakers tab for use with this type of association.

Sequence numbers (0, 1, 2, 3) and (0, 1, 3, 4) represent now-obsolete multi-channel configurations known as *7.1 wide configuration speakers*. These sequences should not be used for new designs, nor should the speaker configuration control panel be used with this setting unless the speaker configuration is truly set up in this fashion.



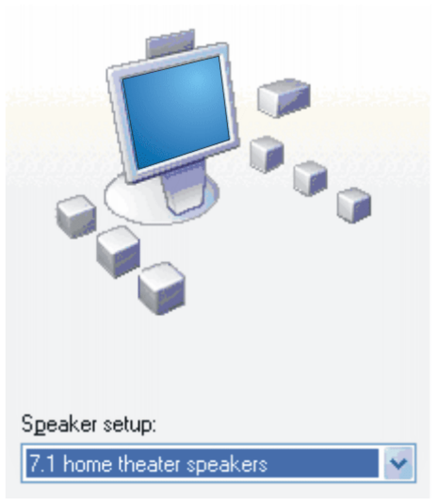

Speaker setup:
7.1 home theater speakers

Figure 4.24 7.1 Home Theater Sequence for a Multi-Channel Association

Table 4.21 details the pin configuration defaults for the 7.1 home theater speaker configuration. Be aware that Microsoft uses the term *Home Theater* to denote a 7.1 configuration, while Dolby† uses *Home Theater* to denote a 5.1 configuration, and *Master Studio* to refer to 7.1.

Sequence (0, 1, 2, 4) is very similar to the recommended sequence of (0, 1, 2) for 5.1 surround. The addition of the grey side surround speaker jacks is the only difference. In sequence (0, 1, 2, 4), sequence 0 is always front (green), sequence 1 is always Center/LFE (orange), sequence 2 is always Rear Surround (black), and sequence 4 is always side surround (grey).

Table 4.21 **Pin Configuration for 7.1 Surround Line Outputs using Sequence (0, 1, 2, 4)**

Connectivity	Loc	Device	Type	Color	Jack Detect	Assoc	Seq
Chassis Jack	Rear	Line Out	3.5 mm	Green	Yes (0x0)	3	0
Chassis Jack	Rear	Line Out	3.5 mm	Orange	Yes (0x0)	3	1
Chassis Jack	Rear	Line Out	3.5 mm	Black	Yes (0x0)	3	2
Chassis Jack	Rear	Line Out	3.5 mm	Grey	Yes (0x0)	3	4

Note: The choice of sequence number determines the function of each port, which must be matched with the proper color for that function. See Appendix A for an example of a completed verb table for this association.

Resource Sharing

UAA guidelines generally discourage sharing of codec resources, such as ADCs and DACs, between multiple pin widgets because they might not always be available. This practice could be a cause of significant user confusion. A DAC or ADC should have an exclusive path to one and only one pin widget. The port controlled by that pin widget should be connected to one and only one jack or integrated device. This exclusivity allows the Microsoft UAA class driver for Intel HD Audio's topology parser to consistently and uniquely identify each audio endpoint.

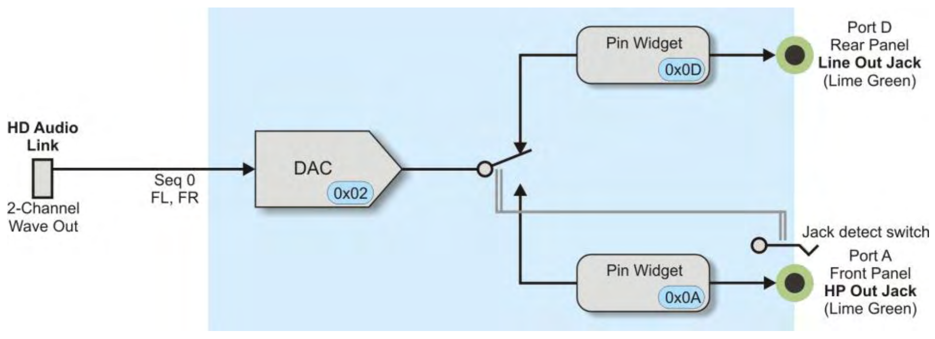
The three exceptions to this rule support usability models that have long been established on the PC. In each case, a single DAC or ADC is associated with multiple pin widgets. Sequence numbers `0xE` and `0xF` are used to indicate this shared usage. While these special cases are not defined in the Intel HD Audio specification, they are part of the requirements for Microsoft's UAA Intel HD Audio class driver compatibility and are consistent with the Intel HD Audio specification.

Redirected Headphone Association

One common usage for headphones is that plugging headphones into the headphone jack disables the built-in speakers or the line-out jacks on the rear panel, as shown in Figure 4.25. In this case, the signal from the DAC at widget ID# 2 would switch from the line-out jack to the headphone-out jack whenever headphones are plugged in, and switch back when the headphones are unplugged, un-muting the line out again.

If a multi-channel configuration is used for line out, like that shown in Table 4.21, then plugging in the headphone would mute all of the multi-channel outputs on the same association. Depending on the driver configuration and what's playing, the headphones may contain only the Front Left and Front Right signals, or they may contain a mix of all the channels that are playing in the stream.

To use this configuration, you must use a DAC which is capable of being routed to either of the two pin widgets. The sequence number for the redirected headphone must always be `0xF`, which indicates to the Microsoft UAA class driver for Intel HD Audio topology parser that a redirected headphone configuration is being requested.



The DAC is routed to only one pin widget at a time. If the headphone jack is plugged in, then the signal from the DAC is routed to the Front Headphone Out and the Rear Line Out is disconnected. If the headphone jack is not plugged in, then the signal is routed to the Rear Line Out and the Front Headphone Out is disconnected.

Figure 4.25 Line Out jack and Headphone Out jack sharing a single DAC in a Redirected Headphone configuration

The redirected headphone configuration is shown in Table 4.22. Both jacks in this case are green, but this use of the same color complies with

Logo requirements because one is on the front and one is on the rear. Both jacks should support jack detection to meet Windows 7 Logo requirements, though the redirected function still works if only the headphone jack supports detection.

Table 4.22 **Pin Configuration for Rear Panel Line Out with Redirected Front Panel HP Out**

Connectivity	Loc	Device	Type	Color	Jack Detect	Assoc	Seq
Chassis Jack	Rear	Line Out	3.5 mm	Green	Yes (0x0)	3	0
Chassis Jack	Front	HP Out	3.5 mm	Green	Yes (0x0)	3	F

Note: See Appendix A for an example of a completed verb table for this association.

A similar configuration is desirable for PCs with a built-in speaker and a headphone jack. It matches the way that TVs and radios with a headphone work, muting the built-in speaker whenever headphones are plugged in. Figure 4.26 shows the block diagram of this configuration.

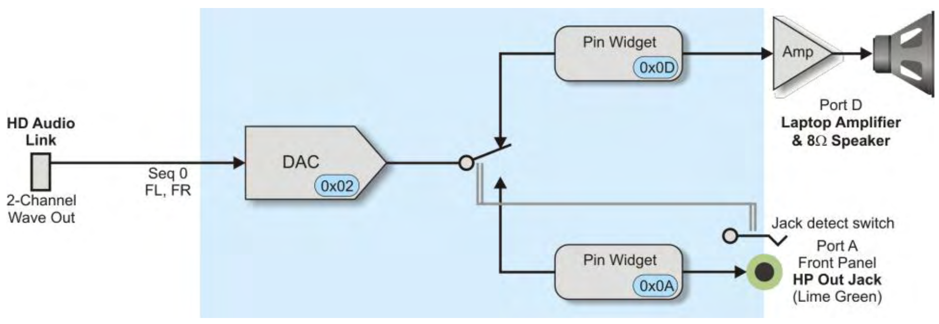


Figure 4.26 Internal Speaker with Redirected Headphone Output

From the codec perspective, the routing and the signal flow in Figure 4.29 are the same as shown in Figure 4.28. However, the pin configuration for the internal speaker is quite different from the line out configuration. No color is specified, and no jack detection is present. The location and connectivity registers are set to a unique value indicating a built-in speaker. Like the previous example, the sequence number for the headphone is set to `0xF` to indicate this behavior. Table 4.23 shows the configuration defaults.

Table 4.23 **Pin Configuration for Internal Speaker with Redirected Front Panel HP Out**

Connectivity	Loc	Device	Type	Color	Jack Detect	Assoc	Seq
Fixed Function	Internal	Speaker	Other Analog	n/a	No (0x1)	1	0
Chassis Jack	Front	HP Out	3.5mm	Green	Yes (0x0)	1	F

Note: See Appendix A for an example of a completed verb table for this association.

Redirected Line Out Association

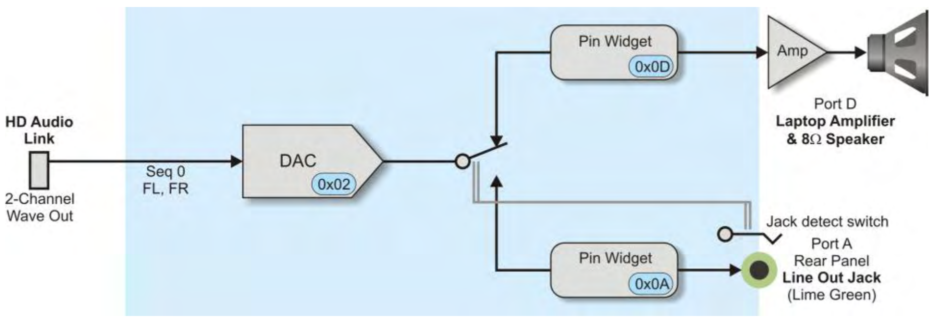


Figure 4.27 Internal Speaker and a Redirected Line Out jack Sharing a Single DAC

This configuration is very similar to the previous configuration. The primary difference is that the green jack is on the rear of the computer, and it is not capable of driving headphones. It is also possible for this configuration to be used with 5.1 or 7.1 associations. To allow for this, the internal speaker always uses Sequence `0xF` in the same way that the headphone output does in the previous examples.

Table 4.24 Pin Configuration for Internal Speaker with Redirected Rear Panel Line Out

Connectivity	Loc	Device	Type	Color	Jack Detect	Assoc	Seq
Fixed Function	Internal	Speaker	Other Analog	n/a	No (0x1)	1	F
Chassis Jack	Rear	LineOut	3.5mm	Green	Yes (0x0)	1	0

ADC Mux Association

Most Intel HD Audio codecs are designed to accept inputs on any of the analog audio ports A thru H, although typically only a few ports are dedicated to input functions. Each ADC usually has a selector that allows it to select one of these input ports to provide input. Figure 4.28 shows an association configured as ADC Mux. It is able to identify an ADC and the ports that ADC can choose to select its signal from.

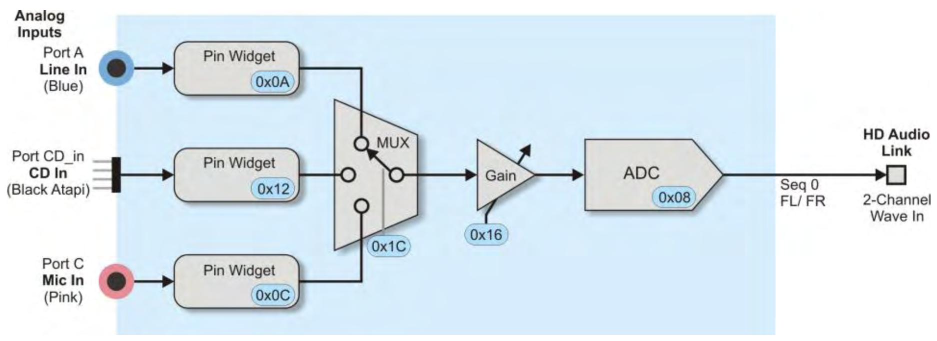


Figure 4.28 Three Pin Widgets Sharing a Single ADC in a Mux Pin Configuration

If there is only one gain control, as shown in the diagram, the OS stores a different gain value for each input to the mux. This feature gives the effect of having three separate gain controls without requiring all three to be implemented in hardware. If there are separate gain controls in each pin widget, then the OS keeps track of each individual volume setting. Table 4.24 shows the configuration defaults for this option.

Table 4.24 **Pin Configuration for Shared Input Mux with Line In, CD, and Microphone In**

Connectivity	Loc	Device	Type	Color	Jack Detect	Assoc	Seq
Chassis Jack	Rear	Line In	3.5 mm	Blue	Yes (0x0)	5	0
Fixed Function	ATAPI	CD In	ATAPI	Black	No (0x1)	5	1
Chassis Jack	Front	Mic In	3.5 mm	Pink	Yes (0x0)	5	E

Note: See Appendix A for an example of a completed verb table for this association.

Even though this example shows how to configure the analog CD analog CD is not recommended, and it might not be supported under Windows

7. The ATAPI connector has no provision for jack detection. If no cable is plugged into the ATAPI connector, set the connectivity to No Connect.

ADC Mix Association

The ADC mix configuration block diagram shown in Figure 4.29 is very similar to the mux configuration. The big difference is that all three inputs are "live" at the same time, which requires independent gain controls for each input in order to adjust the relative level of each input. While allowable under both the Intel HD Audio specification and the UAA guidelines, most users find the mix configuration confusing since it doesn't match the way that A/V receivers are designed. A mix configuration is a poor choice for ease of use because it does not match the behavior of a typical A/V receiver, which uses a Mux to select between inputs. You should avoid using the mix configuration unless you have a specific reason to use it.

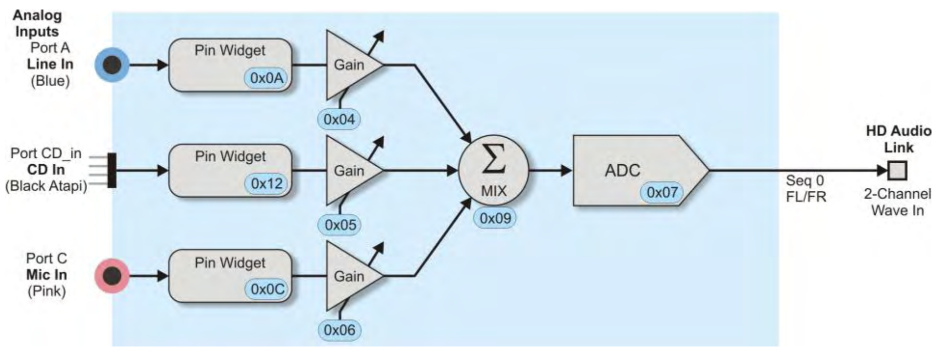


Figure 4.29 Three Pin Widgets Sharing a Single ADC in a Mix Configuration

Since each input port has an independent gain stage, they must have separately addressable widget node IDs, which increases the codec cost and complexity. In terms of the pin configuration shown in Table 4.25, the only difference between a mux and a mix configuration is that the highest sequence number is set to `0xF` rather than `0xE`, signifying mix rather than mux.

Table 4.25 **Pin Configuration for Shared Input Mix with Line In, CD, and Microphone In**

Connectivity	Loc	Device	Type	Color	Jack Detect	Assoc	Seq
Chassis Jack	Rear	Line In	3.5 mm	Blue	Yes (0x0)	5	0
Fixed Function	ATAPI	CD In	ATAPI	Black	No (0x1)	5	1
Chassis Jack	Front	Mic In	3.5 mm	Pink	Yes (0x0)	5	F

Note: See Appendix A for an example of a completed verb table for this association.

These configurations are but a few key ones that can be created from associations and sequences. Links to more detailed information are available at this book's companion Web site.

Resource Allocation

While hardware developers think of the I/O ports as the primary point of contact for the codec, software developers consider the DACs and ADCs in the codec to be the primary point of contact. Intel HD Audio codec designs have a somewhat flexible interconnection between analog I/O ports and resources such as DACs and ADCs that are directly addressed by the software.

The codec must have enough resources in the form of ADCs and DACs to build complete paths for each port, as specified in the default configurations. For instance, if four ports were to be configured as stereo line outputs, but the codec contained only three stereo DACs, the path with the highest association number (and therefore lowest priority) cannot be completed because the driver has run out of DACs to assign.

While most codecs have ADC muxes that can select any of the analog ports, it is not always true of the connections between DACs and ports configured for output. In many designs, a DAC can connect to only one or two ports.

If you are having trouble getting a particular pin configuration to work properly, study the block diagram or widget diagram of the codec to be sure that you have enough DACs and ADCs to fulfill your needs. If you're sure that resources are sufficient, try to isolate the problem by setting all other ports to No Connection. This setting releases any DACs, ADCs, or other elements claimed by these ports.

If your association now works properly with all ports set to No Connection, you have a resource conflict. Re-enable the other associations one at a time until the conflict re-appears, then study the codec block diagram to determine where the conflict is occurring. Or better yet, use colored highlighter markers on the codec block diagram to color code each association. Another approach is to reverse the priority of the associations that are in conflict. If a number of different paths are possible, you can often resolve the issue by causing the resources to be allocated in a different order.

Unsolicited Response

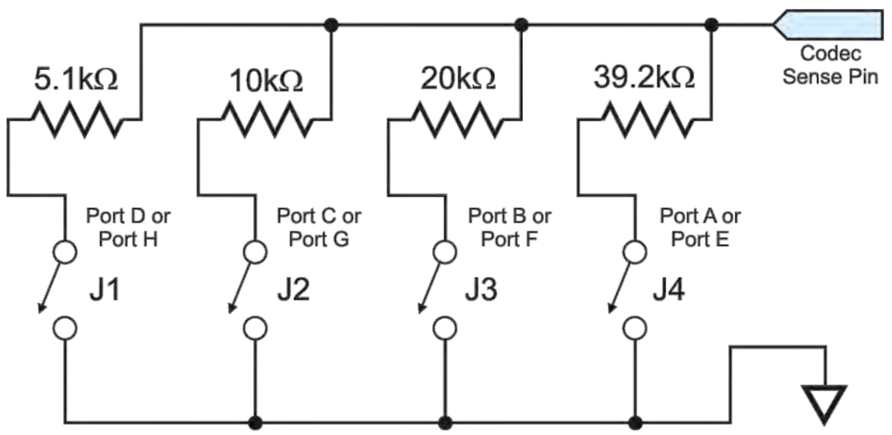
In place of a traditional interrupt request (IRQ), unsolicited responses are messages that are sent by the codec independently of any software request. Typically, they are triggered when the codec's SENSE pins change state because a jack was inserted or removed, but they can also be triggered by such events as when the listener changes a front panel volume control or when the codec latches onto a valid S/PDIF input stream.

The Intel HD Audio bus driver transforms the unsolicited responses into callback events that the function driver registers to receive. The Intel HD Audio bus driver calls the specified callback routine at IRQL DISPATCH_LEVEL, much like a traditional interrupt service routine. The driver can then dispatch events to user-mode applications that have registered to receive callbacks. At the higher levels in the OS, the unsolicited responses may be treated identically to a normal IRQ.

Jack Detection

The Intel HD Audio jack detection mechanism consists of two related but very different functions: jack-insertion detection and impedance sensing. Jack-insertion detection is required by the Windows 7 Logo program, but impedance sensing is optional and is supported only in some codec vendors' function drivers. In some limited cases, impedance sensing can be used to automatically reconfigure a jack to match the analog device that was plugged into the jack.

Jack-insertion detection is accomplished by using the resistor ladder shown in Figure 4.30 to share a single SENSE pin between four ports. The Presence Detect bit in the Pin Sense register inside each pin widget changes to a one whenever the corresponding switch is closed, indicating that a jack has been inserted.



All resistors have a tolerance of 1 percent. Switches J1 thru J4 are closed when a jack is inserted in the associated port. The SENSE_A pin is used for ports A thru D, while SENSE_B is used for ports E thru H. The switches must be isolated from the signal paths of the port.

Figure 4.30 Resistor Stack Connecting Four Jacks to a Single Codec SENSE Pin

It is critical to use resistors with a tolerance of 1 percent or better, and to follow good layout and design practices for these circuits. Try to make sure that all four resistors are made using the same process, so that they track better together over temperature variations. If this circuit does not work properly, it could trigger WHQL failures under the Windows 7 Logo program.

Modifying the Pin Configuration in the BIOS

Often, when bringing up the system, you might find that the BIOS team cannot update the default configurations quickly enough to meet audio test deadlines. If the BIOS chip is mounted in a socket and not encrypted, it is relatively straightforward to modify configuration defaults in the BIOS, by using an external ROM programmer.

Remove the BIOS chip from the motherboard and use the ROM programmer coupled to a PC to read the BIOS contents into a hexadecimal file. Using a hexadecimal editor, search for a verb that you know to be present, such as `0x20A71C70`, which is the first verb in the partial verb table in Figure 4.21. You should be able to see the same verb table patterns that you see in the previous examples: sets of four similar 32-bit words all starting with 2 and followed by the widget ID.

Once you have located the verb table that you wish to edit, use the hex editor to edit the raw data, then blow the updated contents back into the BIOS chip, and put it back into the motherboard.

Most production BIOSes are encrypted, so this technique is unlikely to work with shipping products. However, it could prove to be very handy in meeting tight development schedules.

System Bring-up Trick Using the Microsoft UAA class driver for Intel® HD Audio

In many cases, it may be sufficient to test with the Microsoft UAA class driver for Intel HD Audio. The class driver has an INF file mechanism to transmit a verb table just prior to normal driver initialization. This transmission allows the pin configuration registers to be programmed properly even if the BIOS is not programming them properly.

You can accomplish this transmission by editing a copy of the class driver INF file, HDAUDIO.INF. Be aware that this practice is only suitable for lab work; you cannot submit an edited HDAUDIO.INF as part of a Logo submission package. The mechanism is much like the verb table mechanism, but the surrounding syntax and structure is different. The formation of the verbs themselves is identical.

The InitVerbs registry entry is controlled by the HKR key in the HdAudInit.AddReg section of the HDAUDIO.INF file. Create one entry is for each verb that you want to send to the codec. You also must set a descriptor to indicate the total number of verbs, specified in hexadecimal. Each verb must be preceded by a unique ID number, starting with zero. These unique ID numbers are specified in decimal. An example is shown in Figure 4.31.

Just like verb tables written by the BIOS, you must know the codec ID and the Node ID for each set of commands. The verbs are sent blindly and with no error checking.

```
[HdAudInit.AddReg]
; Number of verbs to be sent to codec
HKR,InitVerbs,NumVerbs,0x00010001,0x00000004
;--Codec ID#2, Port C (NID = 0x0C)
; Set Node ID 0x0C Pin Configs to 0x01813050
HKR,InitVerbs,0000,0x00010001,0x20C71C50 ; Assoc 5, Seq 0
HKR,InitVerbs,0001,0x00010001,0x20C71D30 ; Blue, Jack Detect
HKR,InitVerbs,0002,0x00010001,0x20C71E81 ; Line In, 3.5 mm jack
HKR,InitVerbs,0003,0x00010001,0x20C71F01 ; Rear, chassis
```

This code matches the verb table in Figure 4.22 for a rear line input jack.

Figure 4.31 Verb Table Specified in the INF file of the UAA Class Driver

You can use this feature of the Microsoft UAA class driver for Intel HD Audio to pre-test verb tables before they are provided to the BIOS team or to test various potential configurations of the motherboard. Remember: any changes that you make to the HDAUDIO.INF file are for your own use and may not be submitted to WHQL as part of a Logo package. Instead, any verbs must be written by the BIOS in a production system capable of meeting Logo requirements. You must use an unedited version of the Microsoft UAA class driver for Intel HD Audio INF file to ensure system compliance with Windows 7 Logo requirements.

The examples provided in Appendix A include verb tables in both ASM and INF formats. You can use the examples to build a complete pin configuration for your system.

Display-based Audio

Historically, video and audio signals were always carried by separate physical connectors. With the development of the High Definition Multimedia Interface (HDMI) and, later, Display Port, it become possible for the two types of signals to be carried over a single connector and cable. Supporting these display audio protocols on the PC was challenging, because the rendering of video and audio are handled by separate software stacks that do not communicate with each other.

The video and audio signals are interrelated. For instance, the active video resolution (for example, 1920x1080) affects the amount of bandwidth available for transmitting audio information. Certain video resolutions are incapable of supporting high bit-rate audio streams.

Furthermore, an end user may change the video resolution at any time, potentially rendering a current audio configuration invalid.

As such, it is necessary for the video and audio software stacks to communicate with one another. When Intel architects began designing a solution for supporting HDMI on Intel graphics hardware, we created a hardware-based mailbox for the audio and video drivers to exchange information. This structure was similar to the existing Extended Display Information Data (EDID) commonly found in monitors, yet contained additional information needed for audio. We named the resulting structure EDID-Like Data (ELD). Although the solution was initially unique to Intel products, it has since been standardized in the Intel HD Audio specification.

The ELD contains a header and a baseline block, and may optionally contain vendor-specific data (Figure 4.32).

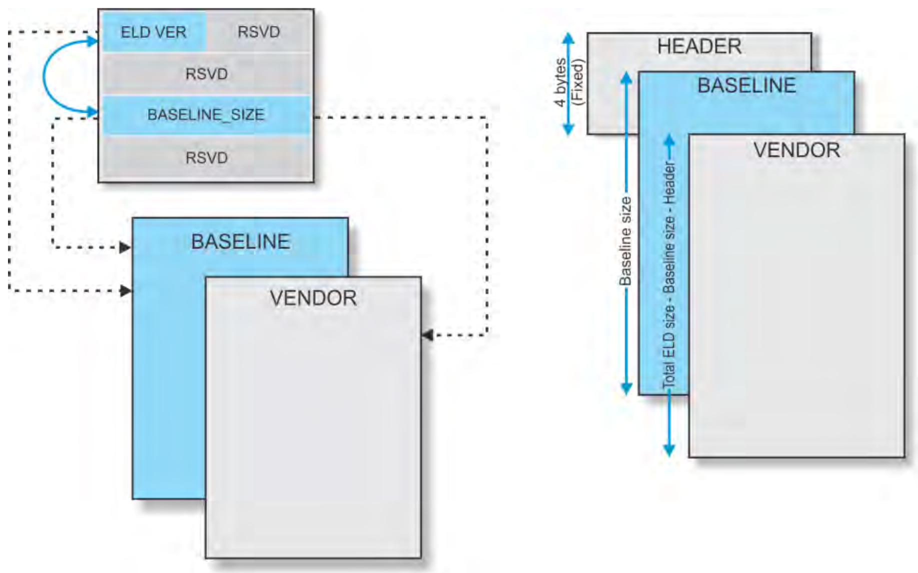


Figure 4.32 ELD Memory Structure

The header is four bytes long and contains two pieces of information: the ELD version and the size of the baseline block. The only currently viable ELD version is version 2, which supports a maximum baseline block size of 80 bytes. The vendor-specific block is implementation-specific, and not required for standard operation.

The baseline block includes the following monitor information:

- User-friendly monitor name string
- Manufacturer name
- Product Code
- Latency of the audio relative to the video. Such latency is typically introduced by performing quality enhancements on the video. Adjusting the audio feed by the latency can prevent A/V synchronization issues.
- HDMI or Display Port connection type
- HDCP support
- A list of supported audio formats stored as CEA Short Audio descriptors
- List of speakers attached to the display (for example, Front Center, Front Left and Right)

The ELD is derived from the attached display's EDID, so if a display is not attached, there can be no ELD. In a typical implementation, the video driver will retrieve the EDID from the monitor and program an ELD for consumption by the audio unit. An ELD is typically generated when a monitor is activated by the video driver; updated ELDs may also be generated when the display resolution is changed.

The audio driver can receive asynchronous notification of the arrival of a new ELD by using the Unsolicited Response Control Verb. Properly configured, the audio codec will generate a new intrinsic unsolicited response when a new ELD is valid. The driver can use the same verb to enable Presence Detect URs, which are triggered whenever a display is attached or removed.

Data Island Packets

In addition to the video and audio sample data, additional information can be sent across the HDMI and Display Port links. This information includes sideband descriptions of the audio/video content and content protection status. This ancillary information is carried over the link in what are known as data island packets (DIP).

The Intel HD Audio specification includes mechanisms for setting the contents of the packets. There are different flavors of packets across HDMI and Display Port, and these can be sent at different frequencies across the links. HDMI codecs must support at least four DIP packet buffers, whereas Display Port codecs only need to support at least one.

The DIP-Size verb allows the drivers to determine the hardware buffer size for the up to eight data island packet buffers. The same verb is also used to determine the size of the EDID-like data.

The DIP buffers can be read and written one byte at a time by using the DIP-Index verb to select the desired buffer and desired byte index, and then using Get/Set DIP-Data verbs to read/write that byte's value. After the Get/Set DIP Data verb is issued, all subsequent Get/Set Data verbs will automatically use the next byte in the same buffer until a new DIP-Index verb is issued.

The DIP-XmitCtrl verb is used to set how often the currently selected packet buffer (as indicated by the last DIP-Index verb) is transmitted over the link. The valid values are:

- Never
- Transmit the packet once
- Transmit the packet at best effort (typically once per video frame)

The DIP buffers can be filled with appropriate information by the audio drivers using the Set DIP-Data verb.

HDCP

Both HDMI and Display Port support a link protection scheme known as High Bandwidth Digital Content Protection (HDCP). HDCP was originally defined on the Digital Visual Interface (DVI), which was incapable of supporting audio. As such, the responsibility for controlling HDCP on Windows operating systems rests with the video drivers. Content protection requests originating from the audio side thus require coordination with the video drivers. Specifically, the video driver is the master of the HDCP state; the audio driver is only allowed to make requests to change state. State changes, if approved, may take time to be activated.

The audio driver can get and set content protection state using the content protection control verbs: GET_CP_CONTROL and SET_CP_CONTROL. GET_CP_CONTROL provides two read-only parameters set by the hardware: Current Encryption State (CES) and Ready. The CES indicates whether HDCP encryption is currently active over the display link. The Ready bit (also called CP_READY in the Intel HD Audio specification) indicates whether or not the hardware is in a state where it is ready to receive encryption state change requests from the audio driver.

The CP_CONTROL verbs also use a read/write parameter titled Content Protection (CP) State. The label is a bit of a misnomer, however. More accurately, the parameter describes the content protection state currently requested by the audio driver.

The three valid states are:

- *Don't care*: The audio driver does not care whether or not HDCP is activated. This is the default state, and allows the video driver to toggle HDCP encryption at the request of video applications without
- *Protection Off*: The audio driver would prefer that HDCP be disabled.
- *Protection On*: The audio driver would prefer that HDCP encryption be active.

Note once again that the Protection On and Off states are just requests from the audio driver. The video driver is under no obligation to honor them. The actual state of HDCP encryption on the display link can only be determined using the CES bit.

When changes to the content protection state are made by the hardware, it will generate a non-intrinsic unsolicited response. This UR contains the CP_READY bit as well as the CP_STATE.

The basic flow for performing audio-side HDCP activation is as follows:

1. The video driver activates the display link. When the link is active, it sets the CP_READY bit in hardware.
2. The audio driver determines it needs to activate link encryption, most likely due to a need to protect premium content. The audio driver uses GET_CP_CONTROL to check CP_READY, which indicates if hardware is ready for content protection operations
3. Assuming CP_READY is true, the audio driver sends a SET_CP_CONTROL with CP State set to Protection On. This verb includes a subtag field that will be used to identify subsequent content protection unsolicited responses.
4. The hardware clears the CP_READY bit, indicating that it is operating on the request. It also responds with a default response, confirming to the audio driver that it has received the request.
5. The audio driver polls CP_READY via GET_CP_CONTROL.

6. The video driver is alerted by the hardware of the request. If HDCP is not already on, it will begin HDCP activation. When complete, the hardware will set the CP_READY bit.
7. Now that CP_READY is set, the hardware will generate the CP UR with the subtag set in step 3.
8. Upon receipt of the CP UR, the audio driver can determine the link's HDCP status by examining the Current Encryption State bit available via GET_CP_CONTROL.

Low Power Enhancements

Design Change Notification (DCN) HDA015B is likely the most far-reaching set of changes between the 1.0 and 1.0A versions of the Intel High Definition Audio 1.0a specification. This DCN outlines significant improvements to low power performance including the ability to control power at the widget level, the ability for jack sense detection to work properly in low-power states, and system wake and reporting of presence, even if the Intel HD Audio link clock is not running when the Intel HD Audio controller is in a low-power state.

The DCN clarifies how to use the previously-unused D1 and D2 power states, and defines a new optional D3cold power state, which allows the lowest possible power consumption. Once a codec is put into D3cold, it can only be awakened by a reset on the Intel HD Audio Link or a double Function Group reset command sequence. Jack detection and other unsolicited responses are disabled while in the D3cold power state.

This DCN also defines an SPDIF keep-alive feature that continues to send audio packets out of the SPDIF output port even when a stream is not in progress. Without this support, most A/V receivers lose roughly the first half-second of each new stream, as they analyze the incoming data to determine what format the data is in before locking on.

If the codec reports that Extended Power State Support (EPSS) is available, then *all* of the low-power features must be implemented by the codec, including new features such as SPDIF Keep Alive and SCMS copy protection bits. There are corresponding WHQL logo tests for almost all of these enhancements, which will only be executed if the codec reports support for EPSS.

The scope of changes in this DCN is quite large, and beyond the scope of this chapter. While all of these changes have been integrated into the 1.0A specification, you can get a better sense of the scope of enhancements by reading the DCN itself, which is available for download from http://www.andrewgrove.org/standards/hdaudio/index.htm.

A closely-related DCN is HDA024-A, which describes the transition from 3.3V to 1.5V signaling on the Intel HD Audio link. Lower voltages help with power management, especially for battery-powered devices. This DCN further defines a "hot attach" mechanism to allow, for example, an Intel HD Audio codec in a dock to be detected and initialized when a laptop is docked in the dock.

Chapter 12

Windows† 7 and Mac OS† X

Virtually all computers sold today incorporate HD Audio, whether Windows or Macintosh-based, and high-quality audio is increasingly more important as form factors become ever smaller. Windows 7 and OS X 10.7 have both continued to refine the audio subsystem.

Both operating systems employ a "class driver"–based approach to supporting audio hardware without third-party drivers. Windows 7 additionally allows third-party drivers provided by the various HD Audio codec vendors, and allows for the class driver functionality to be extended thereby allowing customizations that are equivalent to what can be done with the third-party drivers.

Mac OS X and HD Audio

This section is pretty short and sweet. Apple provides its own audio drivers as part of OS X, to provide a completely sealed system. There is no equivalent of an APO that can be inserted into the HD Audio driver signal-processing chain, though CoreAudio provides a well-implemented audio processing framework at the application level. Independently developed audio plug-ins known as Audio Units can be registered with CoreAudio and applications can use them for processing as desired.

Intel-based Macintoshes have the option of running Windows using Apple's Boot Camp, which takes advantage of the hardware in the Mac. All of the Mac hardware is fully supported under Windows by using the drivers included with Boot Camp.

From the end-user perspective, the audio drivers effectively disappear as a separate entity, which is a plus. Unless you're an engineer working for Apple, there's not much you'll need to know about how HD Audio works on OS X. Regardless, HD Audio codecs must meet both OS X and Windows 7 requirements to be useful on the Mac platform.

Extending the Windows 7 HD Audio Class Driver

Microsoft has a long history of absorbing common functionality into the operating system while at the same time pushing the extensibility and customization to the user interface level. For example, in the transition from Windows 95 to WDM drivers (see Appendix C, "Audio Drivers from DOS to Windows XP"), Microsoft standardized all of the legacy DRV files that had formed the audio drivers and required driver developers for Windows 2000 to use the Windows Driver Model (WDM) miniport for customization and extensibility.

Following suit, under Windows 7 the class driver takes the place of the customized WDM miniport driver. You extend the HD Audio UAA Class driver by using Audio Processing Objects (APOs), in conjunction with Upper Filter Drivers, Lower Filter Drivers, custom property pages, and customized INF files, as shown in Figure 12.1.

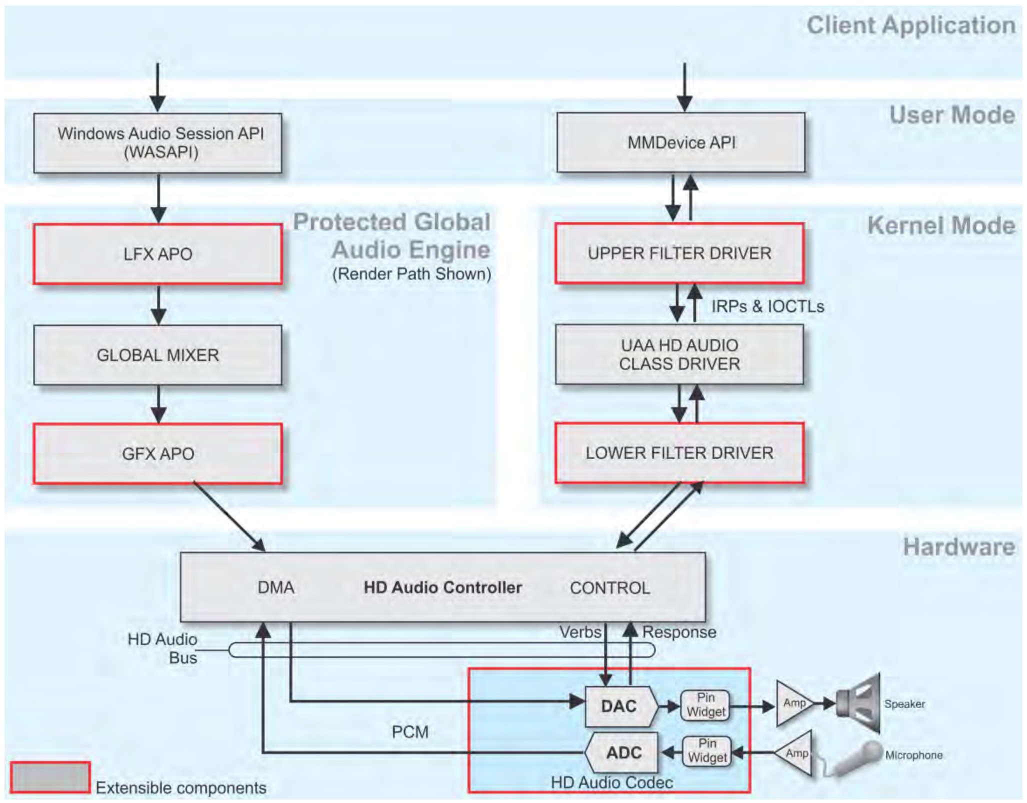


Figure 12.1 Extensible components in the HD Audio driver stack

For Windows 7 (and Vista) the actual audio data packets are no longer passed through the driver stack. Instead, a WaveRT miniport driver connects the output of the global mixer or the GFX directly to the DMA buffers in the HD Audio Controller. The audio function driver sets up the DMA hardware and starts and stops it, but the PCM audio data never passes through the stack.

This means that filter drivers, as well as audio function drivers, cannot be used to perform audio signal processing. Instead an LFX and/or a GFX APO must be used for all signal processing. APOs run in user mode in the Audio DG protected environment.

A filter driver, on the other hand, runs in kernel mode, and is used to manipulate control signals rather than audio data. A lower filter driver can intercept and alter the communications between the HD Audio UAA Class driver and the UAA HD Audio Bus Driver, which in turn controls the HD Audio controller hardware. You can use a lower filter driver to modify verbs and responses passed to and from the HD Audio codec.

An upper filter driver can intercept system audio messages just “above” the HD Audio UAA class driver, to modify the topology of the codec. For example, you can use an upper filter driver to expose the KSPROPERTY_AUDIO_MIC_ARRAY_GEOMETRY property to the rest of the system even though the class driver doesn’t expose this property. This allows you to use the HD Audio UAA Class driver with a microphone array.

An upper filter driver can also expose an interface that applications can call into to control the driver or the hardware. While it is not possible for an application to call into a Lower Filter Driver directly, an Upper Filter Driver can communicate with both an application and a Lower Filter Driver in the same driver stack.

You can use Windows Driver Foundation to create Upper and Lower Filter Drivers. Because they run in Kernel mode, filter drivers must be signed in order to install and run on 64-bit versions of Windows.

A custom property page is the officially sanctioned method for providing a UI for individual audio endpoints. In fact, these property pages are the only place that you can set APO Property Keys; they can’t be set from a normal application. You can create one or more property pages to match the features on the device. The GUI layer is typically built using the aging Win32 API, which was introduced over 15 years ago as part of Windows 95.

Use a customized INF file to load all your extensions along with the HD Audio UAA class driver. The package containing the extensions and the INF file can be submitted for Windows Hardware Device Logo if it passes the all of the tests in the current Windows Logo Toolkit (WLK).

How Custom APOs Interact with System APOs

The Global Audio Engine in Windows 7 is largely composed of system APOs and methods for stringing them together. Most of these APOs are normally invisible from external view (even to a driver programmer) and are strung together automatically based on the input format, the output format, and the formats supported by the LFX and GFX APOs installed on the system. The Microsoft-provided WM audio LFX APO and GFX APO can be replaced by customized versions of each; no others can be replaced.

Table 1.1 System APOs Used by the Windows 7 Audio Engine

System APO Name	Replaceable?	Used For
WM audio LFX APO	Yes	OS-supplied LFX APO, replace with our own
WM audio GFX APO	Yes	OS-supplied GFX APO, replace with our own
CAudioVolume	No	Provides mute and gain control.
CAudioConstrictor	No	Limiting output to 48 kHz, 16-bit
CAudioMixer	No	Mixing multiple audio streams
CAudioRateConvert	No	Sample rate conversion
CAudioRateConvertCMPT	No	Sample rate conversion
CAudioFormatConvert	No	convert int16 to float32, float32 to int32, etc.
CAudioMeter	No	Audio level meters (Peak and RMS)
CAudioMatrix	No	Up or down channel conversion
CAudioLimiter	No	Prevent clipping when rendering
CAudioCopy	No	Capture to several streams at once

These chains of APOs are created dynamically at the start of each new audio stream, and are destroyed at the end of the stream. When the stream is created, the Audio Engine first creates the *Device Pipe* for the endpoint, which contains the GFX APO followed by the limiter, meter, volume control, and format conversion APOs. In the case of encoding GFXs, the GFX is placed last rather than first.

Then the *Stream Pipe* is created, which typically consists of format converter, sample rate converter, LFX APO, volume control, and finally mixer, which outputs to the Device Pipe.

If a second stream is created on the same endpoint, then a new Stream Pipe is created and the output is mixed to the Device Pipe for that endpoint, as shown in Figure 12.2.

The formats for the Device Pipe and for the output of the Stream Pipe are always determined by the settings that the user has selected for sample rate, bit depth, and number of channels.

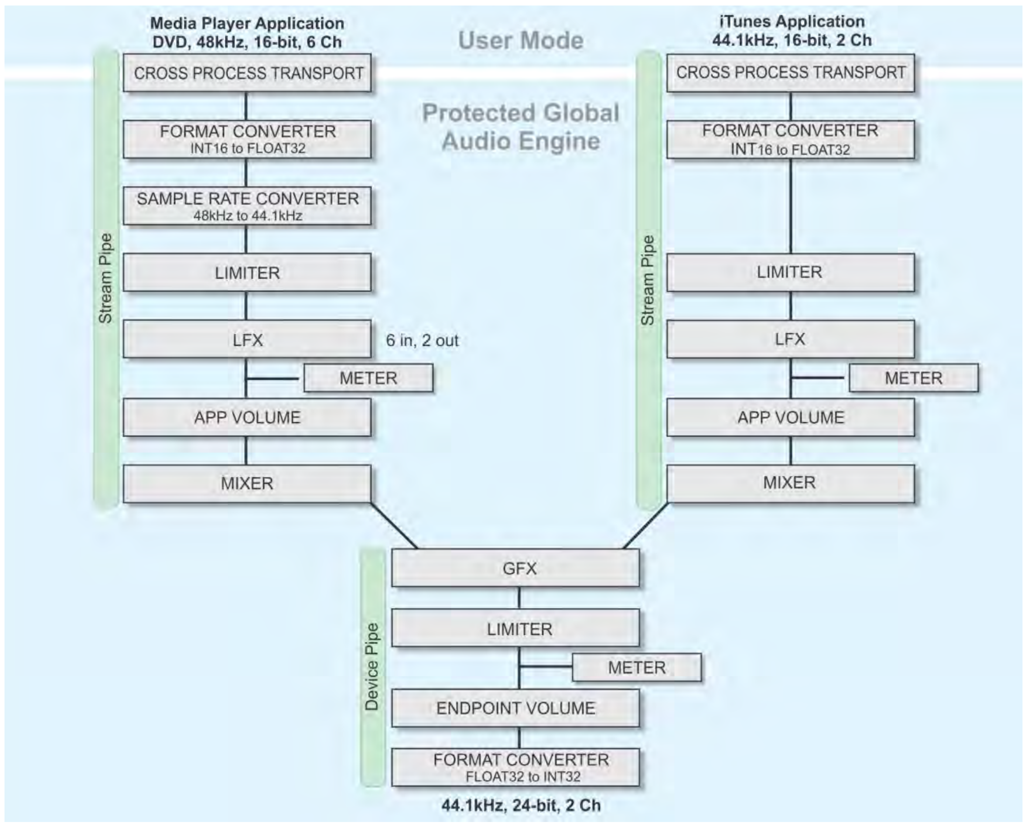


Figure 12.2 Device Pipe and Stream Pipe signal flows

Using a Custom INF File with the Class Driver

Use a custom INF file which references the UAA HD Audio class driver to include the extensible components. The example in Figure 12.3 assigns the CLSIDs of the COM interfaces for the custom LFX, GFX, and UI DLLs. This INF file additionally references the HDAUDIO.INF file for the class driver.

A driver package containing only a custom INF and the extensible components that it references can obtain WHQL logo if it is capable of passing the all of the audio device hardware logo tests. For more information, see the Microsoft white papers entitled “Custom Audio Effects in Windows Vista” and “Reusing Windows Vista Audio System Effects”.

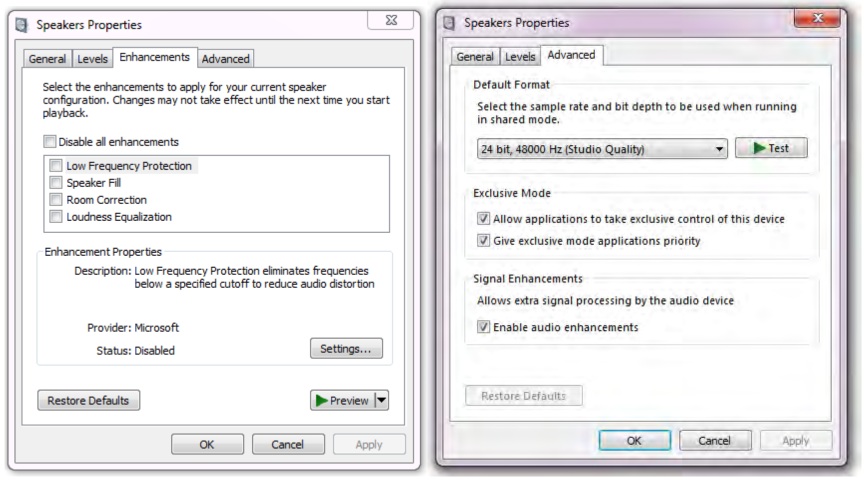
```
[SysFx.AddReg]
HKR,"FX\\0",%PKEY_SYSFX_PreMixClsid%,,%SYSFX_PREMIX_CLSID%
HKR,"FX\\0",%PKEY_SYSFX_PostMixClsid%,,%SYSFX_POSTMIX_CLSID%
HKR,"FX\\0",%PKEY_SYSFX_UiClsid%,,%SYSFX_UI_CLSID%

[Strings]
PKEY_SYSFX_PreMixClsid  = "{D04E05A6-594B-4FB6-A80D-01AF5EED7D1D},1"
PKEY_SYSFX_PostMixClsid = "{D04E05A6-594B-4FB6-A80D-01AF5EED7D1D},2"
PKEY_SYSFX_UiClsid      = "{D04E05A6-594B-4FB6-A80D-01AF5EED7D1D},3
SYSFX_PREMIX_CLSID      = "{B48DEA3F-D962-425a-8D9A-9A5BB37A9904}"
SYSFX_POSTMIX_CLSID     = "{06687E71-F043-403A-BF49-CB591BA6E103}"
SYSFX_UI_CLSID          = "{19166F23-5F08-47F9-BB57-9F57A977D88E}"
```

Figure 12.3 These sections of the INF file specify unique Class ID GUIDs for LFX (PreMix), GFX (PostMix), and PropertyPage (UI).

Replacing the Enhancements Tab

If you choose to replace the default LFX and GFX APOs that ship with Windows 7, it's also a good idea to replace the Enhancements tab that provides controls for the default LFX and GFX. You can replace it with your own property page by specifying it in the INF file, just as you do for the custom LFX and GFX. If you leave the default Enhancements tab in place, it will cause end user confusion. In this case, the only control on the Enhancements property page that will function at all is the "Disable All Enhancements" checkbox. All other controls will appear to function properly, but will not have any effect on anything.

The Disable All Enhancements checkbox is a WHQL Logo test requirement, so this checkbox must be present if you provide your own property page. If you aren't presenting the user with any controls for your LFX or GFX, then you may choose to use no property page at all, by not specifying a property page in the INF file. If you do this, then Windows will create an "Enable audio enhancements" checkbox in the Advanced tab, which is checked on by default (see Figure 12.4).



Speakers Properties

General | Levels | Enhancements | Advanced

Select the enhancements to apply for your current speaker configuration. Changes may not take effect until the next time you start playback.

- [ ] Disable all enhancements
- [ ] Low Frequency Protection
- [ ] Speaker Fill
- [ ] Room Correction
- [ ] Loudness Equalization

Enhancement Properties

Description: Low Frequency Protection eliminates frequencies below a specified cutoff to reduce audio distortion

Provider: Microsoft

Status: Disabled

Settings... | Restore Defaults | Preview | OK | Cancel | Apply

Speakers Properties

General | Levels | Advanced

Default Format

Select the sample rate and bit depth to be used when running in shared mode.

24 bit, 48000 Hz (Studio Quality) | Test

Exclusive Mode

- [✓] Allow applications to take exclusive control of this device
- [✓] Give exclusive mode applications priority

Signal Enhancements

Allows extra signal processing by the audio device

- [✓] Enable audio enhancements

Restore Defaults | OK | Cancel | Apply

Figure 12.4 The default Enhancements tab is shown on the left. The Advanced tab shown on the right adds an "Enable audio enhancements" when a property page is not specified in the INF file.

New Audio Communications Features in Windows 7

Windows 7 includes long-awaited support for an audio communications default device in addition the existing audio "console" default device. Any endpoint can be set as Default Communication Device or Default Console Device, or to be both at the same time.

An application can choose to open the default console device, the Default Communication Device, or a specific device chosen by the application's user. Render and capture default devices are totally independent from each other. There are no formal APIs or system calls for an application to set the default device; the default device(s) can only be set by the user from the Sound dialog box in the Control Panel, or as a result of installing a new device that sets itself to be a default device.

For example, a media player application will typically use the default console render device to play movies or music, while a VoIP application will typically use the default communication device for both render and capture.

Follow-Me Endpoint Heuristics

Each default device type includes a set of heuristics to dynamically switch the default device as devices are added to or removed from the system. For instance, a Bluetooth† headset is considered to be a better match as a communications device than a laptop's built-in microphone and speaker, so pairing a Bluetooth headset will cause it to replace the built-in microphone and speaker as the Default Communication Device, without affecting the Default Console Device. When the Bluetooth headset is disconnected, then the Default Communication Device switches back to the next best match, which in this case is the internal microphone and speaker on the laptop.

In the same manner, connecting the HDMI endpoint of the laptop to a multi-channel audio/video receiver and HDTV will cause the Default Console Render Endpoint to switch to the HDMI render endpoint, and to switch back when the HDMI endpoint is unplugged. The heuristics for the Default Console Device and the Default Communication Device are completely separate. Each uses its own set of weightings and rankings to determine the device priority, but the underlying code is the same for both.

It is possible for OEMs to adjust the heuristics by modifying registry keys. For more details see the Microsoft whitepaper "Default Audio Endpoint Selection in Windows 7" on the MSDN Web site.

Dynamic Stream Redirection and Dynamic Format Change

Prior to Windows 7, if you switch the default endpoint while an application is playing through it, the stream will stop playing and the audio application will usually throw up some kind of error dialog, resulting in a poor user experience, which gets even worse when the follow-me endpoint heuristics cause endpoints to change automatically.

Dynamic stream redirection resolves all of these issues: As long as an application is using DirectSound, DirectShow, Wave, or Media Foundation to play audio, then the global audio engine redirects the stream to the new default endpoint automatically. Applications that call directly into the low-level Windows Audio Session API (WASAPI) or that use a third-party audio stack such as OpenAL will not benefit from stream redirection.

Although the transition from one device to the other sounds almost seamless to the ear, much more is going on behind the scenes. The existing stream is paused while the audio engine filter graph is destroyed and rebuilt to match the new Default Device, and then the stream is restarted. The APOs in the Device Pipe and in the Stream Pipe are configured to use the output sample rate, bit depth, and channel count that the user has configured for the new Default Device. The audio application is unaware that this automatic switching has taken place.

Dynamic Format Change uses the same mechanism when the user switches endpoint formats while a stream is playing and builds a new filter graph on the fly that matches the new format, even if the Default Device hasn't changed.

Dynamic Format Change also means that devices can now inform the OS that their formats have changed without having to unload and reload the driver. A device can now have a hardware sample rate control that implements Dynamic Format Change to rebuild the filter graph dynamically when the sample rate changes.

This is especially important for HDMI and DisplayPort support, which include back-channel control information in the form of EDID-Like Data (ELD) described in the updated Chapter 4. For example, if you switch your A/V receiver from 5.1 to 7.1, then the 6-channel APO Stream Pipe and APO Device Pipe will be destroyed and new 8-channel Stream and Device Pipes will take their place.

Overall this means that in Windows 7 the number of error messages from audio applications is dramatically reduced compared to Windows Vista, resulting in a greatly improved user experience with minimal or no changes to the applications themselves.

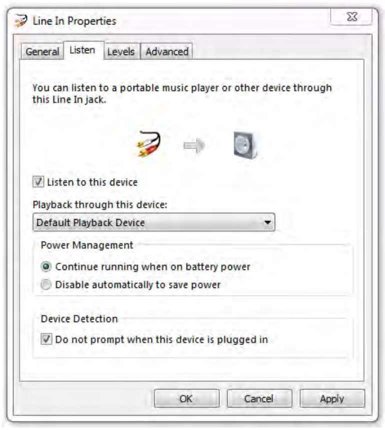
Updated Volume Control Experience

Windows 7 has also greatly refined the Volume Control experience, making it much easier for the end user to access all the different volume controls and sound control panels that might be present on a given system.

It is now possible to configure the mixer panels to show more than one endpoint at a time, and to switch between endpoints without having to close and reopen the mixer panel. Clicking an icon above the slider will usually take you to the associated property page or application.

Capture Monitor

Another long-overdue feature is capture monitor. This feature allows the output of a capture device to be routed directly to the input of a render device. This functionality was virtually impossible to achieve reliably in previous versions of Windows, and allows the user to listen to an MP3 player, for example, through the system's speakers or headphones, as shown in Figure 12.5. Be careful when using this feature with microphone inputs, as it is very easy to create acoustic feedback. It's also possible to configure this setting to allow monitoring only when on line power, and to disable it when on battery power.



You can listen to a portable music player or other device through this Line In jack.

☑ Listen to this device

Playback through this device: Default Playback Device

Power Management

● Continue running when on battery power

○ Disable automatically to save power

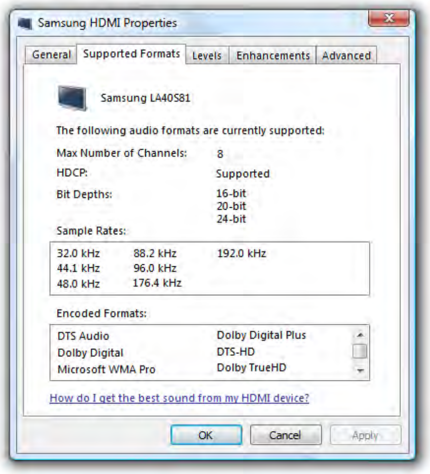
Device Detection

☑ Do not prompt when this device is plugged in

Figure 12.5 The Listen tab for Capture devices controls the Capture Monitor functionality.

Multi-Channel HDMI and DisplayPort support

While it was possible to support multi-channel HDMI in Vista using a third party audio function driver, Windows 7 now adds both HDMI and DisplayPort support to the UAA HD Audio class driver, along with extended format support for high-quality encoded formats, automatic discovery of HDMI sink capabilities, and support for Dynamic Format Change, as shown in Figure 12.6. Jack detection is also supported when plugging in the HDMI connector and the current speaker configuration is now discoverable.



Samsung LA40S81

The following audio formats are currently supported:

Max Number of Channels:	8
HDCP:	Supported
Bit Depths:	16-bit 20-bit 24-bit

Sample Rates:

32.0 kHz	88.2 kHz	192.0 kHz
44.1 kHz	96.0 kHz	
48.0 kHz	176.4 kHz	

Encoded Formats:

DTS Audio	Dolby Digital Plus
Dolby Digital	DTS-HD
Microsoft WMA Pro	Dolby TrueHD

Figure 12.6 The Supported Formats tab of the HDMI endpoint is automatically populated by ELD (EDID-like-data) containing the HDMI device capabilities.

The capabilities shown in Figure 12.6 are the capabilities of the HDMI or DisplayPort sink device at the other end of the cable. As you plug in different HDMI or DisplayPort devices, you will see that the capabilities may change based on the device's characteristics. The name of the final device in the chain will also be shown. If an AV receiver is inserted between the PC and the display device, then its capabilities will also be included. For example, the Samsung TV shown above is only capable of 2 channels when plugged directly into the PC. The 8-channel capability and the ability to decode premium formats such as Dolby TrueHD and DTS-HD are provided by the AV receiver, not by the TV.

Exclusive Mode

Much of the previous discussion has been focused on Shared Mode, which is the typical usage for most endpoints. In Shared Mode, multiple audio streams going to the same endpoint are mixed together, as shown in Figure 12.2.

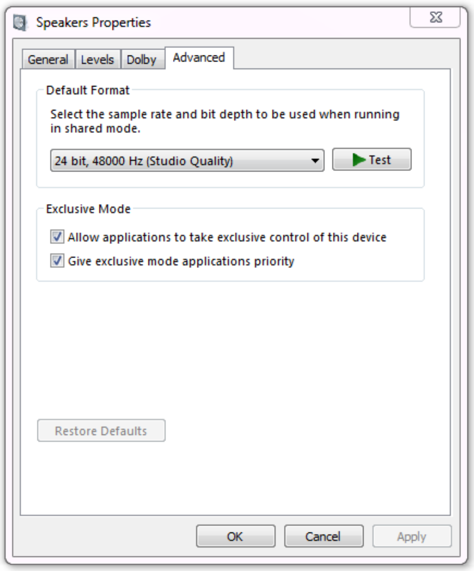
Exclusive Mode does not mix any audio together; in fact it doesn't even allow APOs to be inserted into the signal chain. The choice of Exclusive Mode or Shared Mode is determined by the application and by the audio format. For example, a Blu-ray† player application might choose to pass a signal encoded with Dolby† TrueHD directly to an AV receiver, which will decode the signal and pass it on to the speakers attached to the receiver.

In this case, there would be no volume control on the PC for this endpoint, because the encoded TrueHD stream cannot be modified by either hardware or software without corrupting the data. The only way to modify a TrueHD stream, for example, is to decode it and then modify or scale the un-encoded PCM signal.

Pro audio applications such as ProTools† or Cakewalk†, which contain built-in mixing capabilities, will typically use Exclusive Mode to guarantee that the signals leaving the application will be unmodified by APOs associated with the endpoint.

You can determine Exclusive Mode behavior for each endpoint in the Advanced tab. Figure 12.7 shows the default settings for each endpoint. You can totally block the use of Exclusive Mode by unchecking "Allow applications to take exclusive control of this device". In this case, only Shared Mode will be available, and applications that try to open in Exclusive Mode will fail.

If you uncheck the "Give exclusive mode applications priority" checkbox, then an application that attempts to open the endpoint in Exclusive Mode will fail if that endpoint is currently open in Shared Mode. If this checkbox is checked, then the Shared Mode session will be destroyed if another application opens the endpoint in Exclusive mode.



Speakers Properties

General | Levels | Dolby | Advanced

Default Format

Select the sample rate and bit depth to be used when running in shared mode.

24 bit, 48000 Hz (Studio Quality)

Exclusive Mode

☑ Allow applications to take exclusive control of this device

☑ Give exclusive mode applications priority

Figure 12.7 You can control Exclusive Mode behavior in the Advanced properties tab for the current audio endpoint.

Here's an interesting example of Exclusive Mode in action: Premium versions of Windows 7 and Windows Vista include a DirectShow Dolby Digital decoder which is capable of decoding a 5.1 AC3 bitstream into 6 channels of PCM. PCs that support SPDIF out are capable of playing the un-decoded 5.1 bitstream directly to the SPDIF output to be decoded by the AV receiver at the far end of the SPDIF cable by placing the endpoint into Exclusive Mode. In this case, none of the APOs or volume controls will have any effect because the system is in Exclusive Mode. However, if you uncheck "Allow applications to take exclusive control of this device", then the Windows Media Player application will not be able to open the endpoint in Exclusive Mode. Windows Media Player will instead insert the built-in Dolby Digital decoder into the application signal chain, and will output 5.1 PCM to the endpoint in Shared Mode. If the endpoint has a GFX that includes a real-time encoder such as Dolby Digital Live or DTS Interactive, then the 5.1 PCM signal will be processed by any LFX APOs associated with that endpoint, and then will be recoded to a bitstream format that is transmitted to an AV receiver over the SPDIF cable. The AV receiver will then decode this final bit stream to provide discrete 5.1 surround.

Low Power Enhancements

The low-power updates to the HD Audio 1.0a specification are supported by Windows 7. There is support for PortCls idle power management, which monitors the time that each audio device has been idle and determines whether to power down some or all of the codec's analog circuitry based on the system power management state

There is also improved granularity for how different parts of the codec can be powered down. A new IAdapterPowerManagement2 API has been added that allows drivers to distinguish system power state changes from a PortCls idle power management timeout and that also provides a way for an audio function driver to opt out of PortCls idle power management.

Virtualized 8253 Timer

Back in the days of DOS, sounds on a PC were typically generated by an 8253 timer chip, which generated a square wave that was routed to an internal speaker. This circuitry has been part of a PC ever since, though in recent years most computers designed for a 64-bit OS have not implemented it, even though this is a primary method of providing audio feedback for unsighted users. Take a look at the section of Chapter 7 titled "Sticky Keys, Toggle Keys, and the 8253 Timer" for more detail.

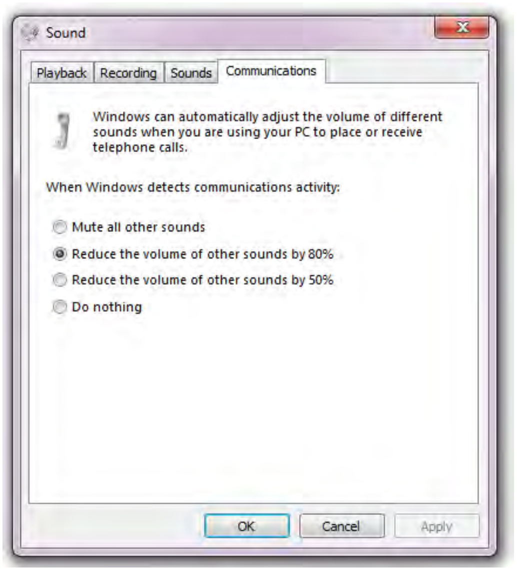
An unfortunate side-effect of this implementation was that an analog mixing path was necessary to mix this square wave signal into system outputs whenever a sound was generated. This meant that some analog portions of the HD Audio codec always needed to be powered up, even when the system wasn't generating any signals, because the OS had no knowledge of when these tones might be generated.

In order to be able to power down the analog section of the codec when not playing sound yet ensure that accessibility sounds will continue to be heard, Windows 7 implements a virtual 8253 timer. The Beep() API was rewritten to call a new software tone generator that responds to the same controls as the 8253 timer but outputs a PCM signal to the Default Console Render Device. This in turn allows the codec to enter D3 sleep state when the codec is idle, but still guarantees that all of the Beep() sounds will be heard by the user, at least if the master volume and mute controls are turned on.

System-wide Attenuation or Ducking

Radio broadcasters have long used the term *ducking* to describe the process of turning down the music when the DJ is speaking. Now Windows 7 can do the same thing when the Default Communications Device is used for a VoIP call.

The idea is that when the Default Communications Device is in use, then any other render endpoints can be automatically turned down or muted while the communication stream is active, as shown in Figure 12.8.



Windows can automatically adjust the volume of different sounds when you are using your PC to place or receive telephone calls.

When Windows detects communications activity:

- Mute all other sounds
- [✓ Reduce the volume of other sounds by 80%]
- Reduce the volume of other sounds by 50%
- Do nothing

Figure 12.8 The new Communications tab in the Sound dialog box of the Control Panel allows the user to globally configure the automatic ducking behavior.

Applications written to take advantage of the IAudioVolumeDuckNotification interface can customize or override the default ducking behavior. For instance, a media player application could register for ducking notifications, pause when the communication stream is started, and then resume playback once the communication stream has been closed.

Event-Driven WaveRT

Under Windows Vista, audio clients that ran in Push Mode had to repeatedly poll the size of the available buffer to decide when to write data to or read data from an audio buffer, which resulted in latencies that were unsuitable for real-time communications applications.

Event-driven mode, also known as Pull Mode, is a Windows 7 logo requirement that significantly reduces overall audio latencies and minimizes audio drop-outs. A driver package must add the `PKEY_AudioEndpoint_Supports_EventDriven_Mode` property key to the INF file to meet the logo requirement.

From a hardware perspective, Pull Mode is similar to double-buffered DMA transfers, where the interrupt is fired at the halfway point and at the end of each allocated buffer, while Push Mode uses a timer to wake up and see if enough time has elapsed to do something useful with the audio.

HDCP and SCMS Copy Protection Support

Applications can now call into the IMFOutputPolicy interface and the IMFTrustedOutput interface while a stream is playing to set the SCMS and/or HDCP copy protection states or to disable digital output on a digital output stream. To play the protected content, the underlying audio driver must be a fully-signed trusted driver; that is, the driver must be logo-certified for DRMLevel of 1300 or higher. Use of a test-signed driver is not sufficient.

Windows Media Player OCX Device Selection

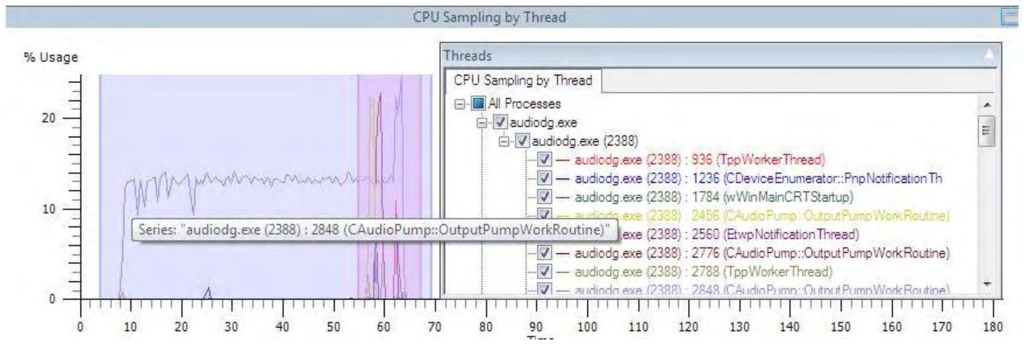
Applications that call into the Windows Media Player API to play media files can use the OCX device selection feature to select a specific render endpoint that is typically selected by the user. Prior to Windows 7, Media Player always chose the Default Console Render Endpoint and there was no way to override this behavior.

Windows Performance Analysis Tools

The Windows 7 SDK includes XPerf.exe, XPerfView.exe, and XBootMgr.exe as the Windows Performance Analysis Tools that are built on top of the Event Tracing for Windows (ETW) infrastructure that is part of Windows 7.

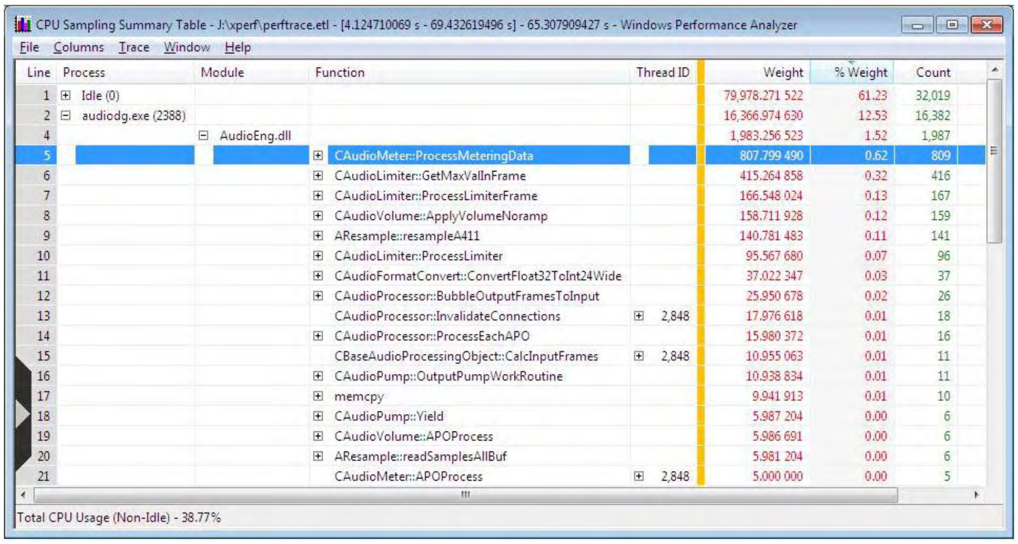
XPerf.exe is used to captures traces, post-process them for viewing on any machine, and to perform command-line (action-based) trace analysis. XBootMgr.exe is used to automate on/off state transitions and captures traces during these transitions. XPerfView is used to view and analyze traces created by the other two tools.

You gather logs by running XPerf.exe from the command line, and then stop it when you want to stop sampling. It creates an .ETL file containing all of the data gathered during the session. XPerfView is then used to analyze and display the data that was gathered, as shown in Figure 12.9 and Figure 12.10.



```
Series: "audiodg.exe (2388) : 2848 (CAudioPump::OutputPumpWorkRoutine)"

All Processes
  audiodg.exe
    audiodg.exe (2388)
      — audiodg.exe (2388) : 936 (TppWorkerThread)
      — audiodg.exe (2388) : 1236 (CDeviceEnumerator::PnpNotificationTh
      — audiodg.exe (2388) : 1784 (wWinMainCRTStartup)
      — audiodg.exe (2388) : 2456 (CAudioPump::OutputPumpWorkRoutine)
      .exe (2388) : 2560 (EtwpNotificationThread)
      — audiodg.exe (2388) : 2776 (CAudioPump::OutputPumpWorkRoutine)
      — audiodg.exe (2388) : 2788 (TppWorkerThread)
      — audiodg.exe (2388) : 2848 (CAudioPump::OutputPumpWorkRoutine)
```

Figure 12.9 XPerfView.exe output showing a worker thread using about 14 percent of the CPU's capacity



CPU Sampling Summary Table - J:\xperf\perftrace.etl - [4.124710069 s - 69.432619496 s] - 65.307909427 s - Windows Performance Analyzer

File Columns Trace Window Help

Line	Process	Module	Function	Thread ID	Weight	% Weight	Count
1	⊞ Idle (0)				79,978.271 522	61.23	32,019
2	⊟ audiodg.exe (2388)				16,366.974 630	12.53	16,382
4		⊟ AudioEng.dll			1,983.256 523	1.52	1,987
5			⊞ CAudioMeter::ProcessMeteringData		807.799 490	0.62	809
6			⊞ CAudioLimiter::GetMaxValInFrame		415.264 858	0.32	416
7			⊞ CAudioLimiter::ProcessLimiterFrame		166.548 024	0.13	167
8			⊞ CAudioVolume::ApplyVolumeNoramp		158.711 928	0.12	159
9			⊞ AResample::resampleA411		140.781 483	0.11	141
10			⊞ CAudioLimiter::ProcessLimiter		95.567 680	0.07	96
11			⊞ CAudioFormatConvert::ConvertFloat32ToInt24Wide		37.022 347	0.03	37
12			⊞ CAudioProcessor::BubbleOutputFramesToInput		25.950 678	0.02	26
13			CAudioProcessor::InvalidateConnections	⊞ 2,848	17.976 618	0.01	18
14			⊞ CAudioProcessor::ProcessEachAPO		15.980 372	0.01	16
15			CBaseAudioProcessingObject::CalcInputFrames	⊞ 2,848	10.955 063	0.01	11
16			⊞ CAudioPump::OutputPumpWorkRoutine		10.938 834	0.01	11
17			⊞ memcpy		9.941 913	0.01	10
18			⊞ CAudioPump::Yield		5.987 204	0.00	6
19			⊞ CAudioVolume::APOProcess		5.986 691	0.00	6
20			⊞ AResample::readSamplesAllBuf		5.981 204	0.00	6
21			CAudioMeter::APOProcess	⊞ 2,848	5.000 000	0.00	5

Total CPU Usage (Non-Idle) - 38.77%

Figure 12.10 XPerfView.exe Sampling Summary Table of the same data from Figure 12.9, showing breakdown of audio processing by module.

The CPU Sampling by Thread view is a great way to see which subroutines are getting called over time, and the sampling summary table is great way to see how much time is being spent in each subroutine.

You can also use XBootMgr.exe to automate and gather performance logs during system boot. This is one of the few ways to have visibility into startup performance and this can provide significant insight into what's going on.

Finally, powerful WPF Performance tools are also installed as part of the toolkit. If you create GUI applications using WPF, this is a great way to verify that performance is up to par.

Echo Cancellation and Microphone Arrays

There is no single agreed-upon way to implement echo cancellation and array microphones on a PC. Almost all communications applications implement some sort of echo cancellation in the application, and a few also perform some type of array microphone processing. Likewise almost all OEMs implement echo cancellation inside the driver stack of most models, and some models also perform some type of array microphone processing in the driver stack, typically inside an APO.

This means that there will be cases where two echo cancellers are running at the same time, with different characteristics. Sometimes this works out just fine, other times it can be a train wreck.

The argument for driver-based echo cancellers is that they are very close to the hardware, and therefore have the lowest latency. Also, in many cases the capture endpoint and the render endpoint are on the same device, and can be implemented to take advantage of each other. For instance, it is possible to tune a built-in microphone and built-in speakers to perform well with each other.

This model breaks down a bit when you consider that Windows audio doesn't have any implicit associations between capture and render devices, thus there is no guarantee that the user will have actually selected the built-in microphone as the Default Communications Capture Endpoint and selected the built-in speakers as the Default Communications Render Endpoint.

For example, if you are using the built-in microphone and speakers for a communications session, and then plug in a set of headphones, then the headphones will become the Default Communications Render Endpoint. Any driver-based echo cancellation going on between the microphone and speakers is no longer needed, but is not necessarily removed from the signal path.

Microsoft therefore recommends that the application perform the echo cancellation and microphone array processing, and that's exactly what they do in Office Communicator 2007 and its successor, Microsoft Lync. Microsoft also ships a DirectX Media Object (DMO) with Windows 7 to provide this same functionality for other applications.

The Microsoft Kinect SDK for Windows 7 provides an updated version of this DMO that takes full advantage of the four array microphones in the Kinect. Although the Kinect connects via USB, rather than HD Audio, it's a good example of how to implement a four-microphone array.

When things aren't working well together, it's sometimes difficult to determine whether the problem is with the echo canceller or with the network. If you are experiencing network latencies, this can sometimes sound like the echo canceller is misbehaving. Therefore it's important to be able to distinguish between acoustical issues and network issues.

If you are testing the communications capabilities of systems under design, it's always best to establish a golden system that works well in all cases, and then test against that. You can accomplish this by using a well-provisioned local network in isolation from the Internet, and use it to connect two acoustically isolated test locations, preferably close to each other.

For your two baseline microphones, consider using a professional grade "shotgun" microphone, such as a Rode NTG-1 or an Audio Technica AT-875R connected through an XLR-to-USB adapter that provides phantom power for the microphone. Shure, MXL, Blue, and Centrance are among vendors who make these adapters. Mount the microphone on a stand that is not mechanically attached to the speakers. You should be able to stand up to 6 feet away from the microphone without applying any additional signal processing.

For speakers, consider using Avantone Active Mixcubes or equivalent full-range powered monophonic speakers attached to the left channel of the render endpoint.

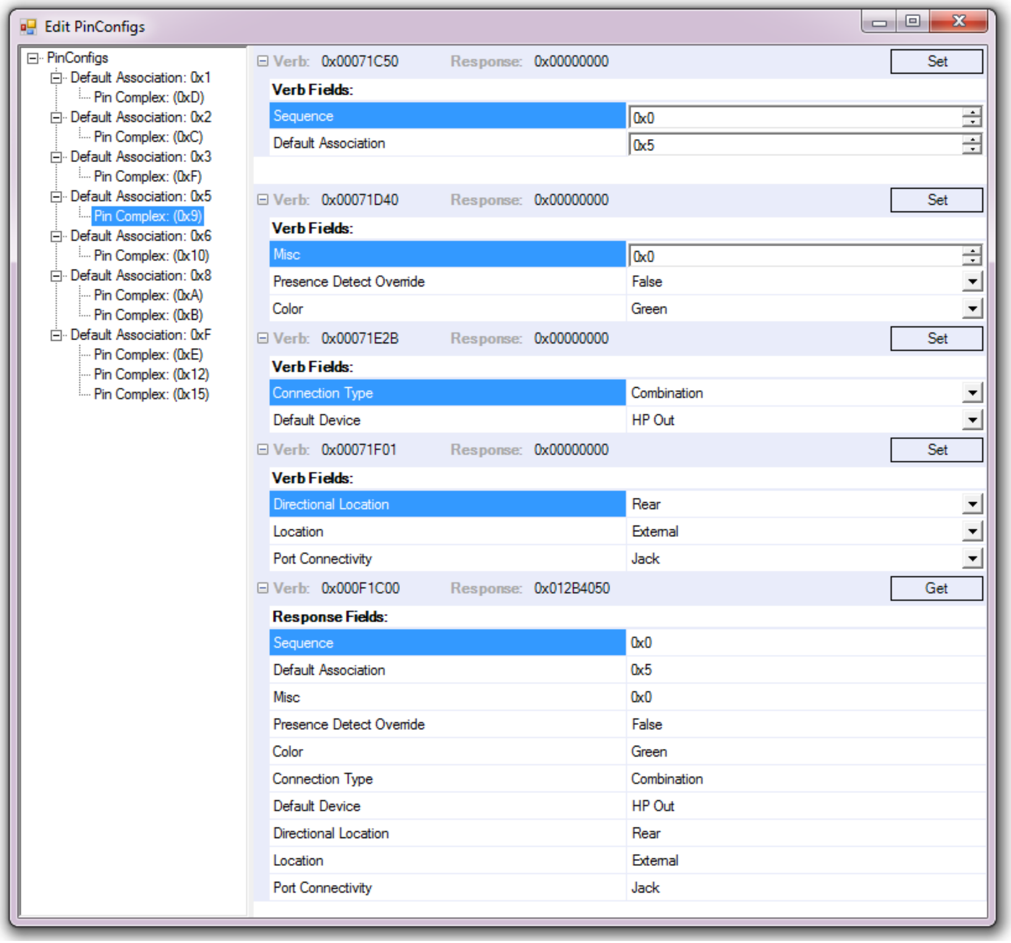
Set up and qualify your baseline system so that it is working well, and then compare the systems under design to this golden reference. This should make it easier to evaluate new software components without changing hardware, and vice versa.

HDAU - HD Audio Tool

Windows 7's release was accompanied by a completely updated High Definition Audio Tool (HDAU) that can be used to create or modify the verb tables that are stored in the BIOS, and that are used to populate the Pin Configuration registers whenever power is restored to the codec.

Previous versions of HDAU performed all processing in software, which required the tool to acquire and maintain an accurate model of the HD Audio codec. The current version actually runs on the audio codec hardware, so when you change a setting, you are changing it in real-time, and the codec itself simply ignores any invalid commands.

Also, it's now possible to temporarily edit a system's PinConfigOverrides in the registry. These registry settings are used in place of the BIOS settings if present. To do this, you must locate the HDAU.exe file and launch it with administrator privileges, as shown in Figure 12.11 and Figure 12.12.



- PinConfigs
  - Default Association: 0x1
    - Pin Complex: (0xD)
  - Default Association: 0x2
    - Pin Complex: (0xC)
  - Default Association: 0x3
    - Pin Complex: (0xF)
  - Default Association: 0x5
    - Pin Complex: (0x9)
  - Default Association: 0x6
    - Pin Complex: (0x10)
  - Default Association: 0x8
    - Pin Complex: (0xA)
    - Pin Complex: (0xB)
  - Default Association: 0xF
    - Pin Complex: (0xE)
    - Pin Complex: (0x12)
    - Pin Complex: (0x15)

Verb: 0x00071C50 Response: 0x00000000 Set

Verb Fields:

Sequence	0x0
Default Association	0x5

Verb: 0x00071D40 Response: 0x00000000 Set

Verb Fields:

Misc	0x0
Presence Detect Override	False
Color	Green

Verb: 0x00071E2B Response: 0x00000000 Set

Verb Fields:

Connection Type	Combination
Default Device	HP Out

Verb: 0x00071F01 Response: 0x00000000 Set

Verb Fields:

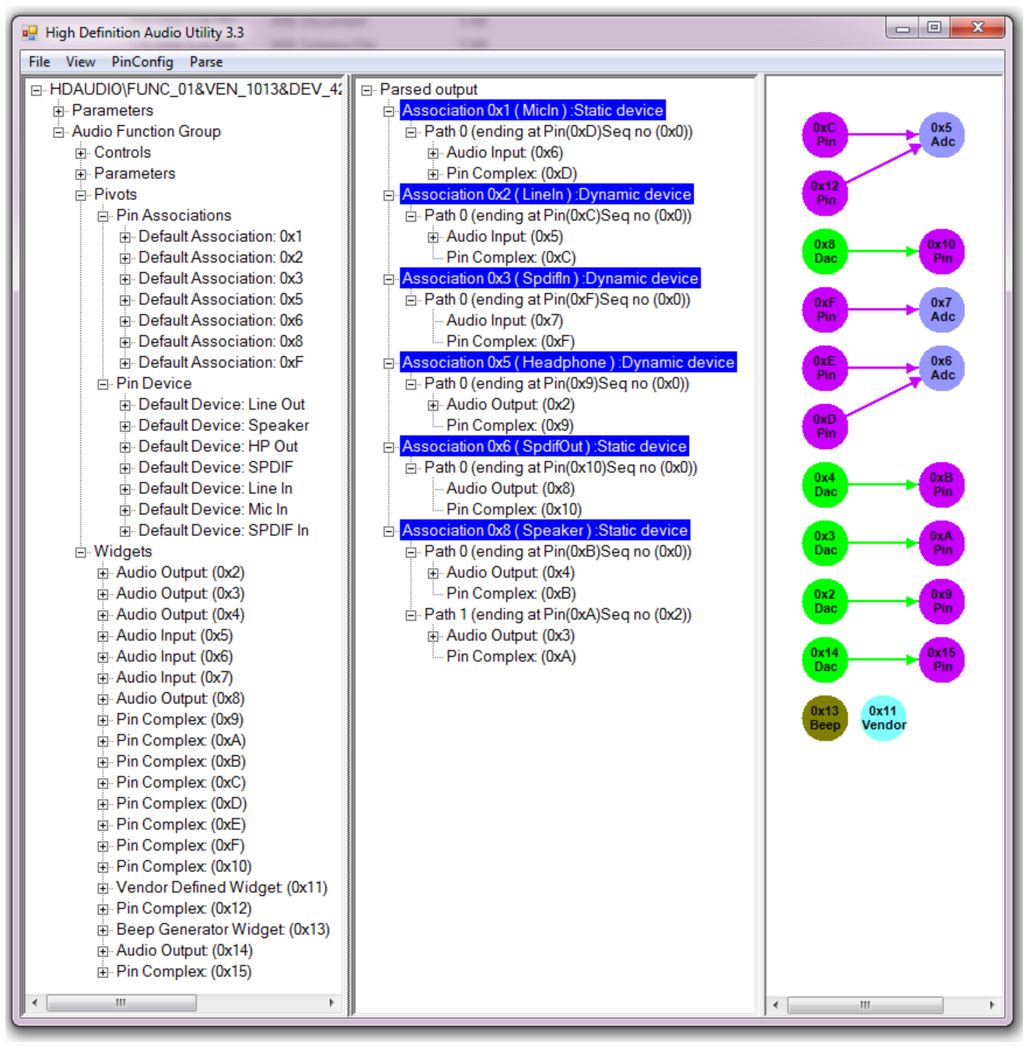
Directional Location	Rear
Location	External
Port Connectivity	Jack

Verb: 0x000F1C00 Response: 0x012B4050 Get

Response Fields:

Sequence	0x0
Default Association	0x5
Misc	0x0
Presence Detect Override	False
Color	Green
Connection Type	Combination
Default Device	HP Out
Directional Location	Rear
Location	External
Port Connectivity	Jack

Figure 12.11 The HDAU Edit Pin Configs dialog box. Select new values in the right-most column, and then click Set to send them to the codec.



```
⊟ HDAUDIO\FUNC_01&VEN_1013&DEV_42
    ⊞ Parameters
    ⊟ Audio Function Group
        ⊞ Controls
        ⊞ Parameters
        ⊟ Pivots
            ⊟ Pin Associations
                ⊞ Default Association: 0x1
                ⊞ Default Association: 0x2
                ⊞ Default Association: 0x3
                ⊞ Default Association: 0x5
                ⊞ Default Association: 0x6
                ⊞ Default Association: 0x8
                ⊞ Default Association: 0xF
            ⊟ Pin Device
                ⊞ Default Device: Line Out
                ⊞ Default Device: Speaker
                ⊞ Default Device: HP Out
                ⊞ Default Device: SPDIF
                ⊞ Default Device: Line In
                ⊞ Default Device: Mic In
                ⊞ Default Device: SPDIF In
        ⊟ Widgets
            ⊞ Audio Output (0x2)
            ⊞ Audio Output (0x3)
            ⊞ Audio Output (0x4)
            ⊞ Audio Input (0x5)
            ⊞ Audio Input (0x6)
            ⊞ Audio Input (0x7)
            ⊞ Audio Output (0x8)
            ⊞ Pin Complex (0x9)
            ⊞ Pin Complex (0xA)
            ⊞ Pin Complex (0xB)
            ⊞ Pin Complex (0xC)
            ⊞ Pin Complex (0xD)
            ⊞ Pin Complex (0xE)
            ⊞ Pin Complex (0xF)
            ⊞ Pin Complex (0x10)
            ⊞ Vendor Defined Widget (0x11)
            ⊞ Pin Complex (0x12)
            ⊞ Beep Generator Widget (0x13)
            ⊞ Audio Output (0x14)
            ⊞ Pin Complex (0x15)
```

```
⊟ Parsed output
    ⊟ Association 0x1 ( MicIn ) :Static device
        ⊟ Path 0 (ending at Pin(0xD)Seq no (0x0))
            ⊞ Audio Input (0x6)
            ⊞ Pin Complex (0xD)
    ⊟ Association 0x2 ( LineIn ) :Dynamic device
        ⊟ Path 0 (ending at Pin(0xC)Seq no (0x0))
            ⊞ Audio Input (0x5)
              Pin Complex (0xC)
    ⊟ Association 0x3 ( SpdifIn ) :Dynamic device
        ⊟ Path 0 (ending at Pin(0xF)Seq no (0x0))
              Audio Input (0x7)
              Pin Complex (0xF)
    ⊟ Association 0x5 ( Headphone ) :Dynamic device
        ⊟ Path 0 (ending at Pin(0x9)Seq no (0x0))
            ⊞ Audio Output (0x2)
              Pin Complex (0x9)
    ⊟ Association 0x6 ( SpdifOut ) :Static device
        ⊟ Path 0 (ending at Pin(0x10)Seq no (0x0))
              Audio Output (0x8)
              Pin Complex (0x10)
    ⊟ Association 0x8 ( Speaker ) :Static device
        ⊟ Path 0 (ending at Pin(0xB)Seq no (0x0))
            ⊞ Audio Output (0x4)
              Pin Complex (0xB)
        ⊟ Path 1 (ending at Pin(0xA)Seq no (0x2))
            ⊞ Audio Output (0x3)
              Pin Complex (0xA)
```

Figure 12.12 The Microsoft HD Audio Configuration Utility main screen, with parsing results on the right.

For more information and to download the tool, search Microsoft's MSDN Web site for "High Definition Audio Tool".

Windows 7 Driver Installation

Windows Vista introduced the concept of a Driver Store, a storage area consistent of all the drivers currently available on that system, regardless of whether they are currently installed or not. For instance, an OEM could choose to pre-install the drivers for an optional USB headset that might be bundled with the system. By pre-installing into the Driver Store, the driver is immediately available when the headset is plugged in for the first. The driver is only loaded when the hardware is actually detected.

You can use PNPUtil.exe to view and edit the driver store. To see all the drivers in the Driver Store, for instance, open a command prompt and type

```
PNPUtil -e
```

To remove an item from the Driver Store, you must determine the name of the INF that you wish to remove, then open a command prompt with administrator privileges, then type

```
PNPUtil -d OEM58.INF
```

As part of being installed into the Driver Store, the name of the INF file is changed to the form OEM*xx*.INF, where the *xx* represents a unique ID within the driver store. The Driver Store uses the date and version specified in each INF file to determine which one is the newest INF file for a particular device. If the Plug and Play ID and the version and date of the INF file being installed are the same as a previously installed INF file, then the new INF file is ignored and nothing is added to the Driver Store.

During development and testing, it is very easy to end up with many different versions of your INF file in the driver store, and this can cause unpredictable behavior if you are not careful. The best strategy is to ensure that only one instance of your driver package under development is stored in the Driver Store. You can do this by using PnPUtil.exe to view the list of items in the Driver Store, and then manually removing any items that you see fit. However, there are easier ways to accomplish this.

How to Install/Remove Drivers Using Device Manager

If you will be installing and uninstalling drivers on a regular basis, you should drag the Device Manager control panel to your Start menu so that it is readily available. In Device Manager, expand the entry for "Sound, video, and game controllers" to see the audio devices that are installed.

Right-click the device you wish to update and then select “Update Driver Software” from the drop-down menu. In the next dialog, select the lower choice: “Browse my computer for driver software”. In the next dialog after that, again select the lower choice: “Let me pick from a list of device drivers on my computer”. This will bring up yet another dialog that shows all of the drivers in the Driver Store that currently match the device.

Click the “Have Disk…” button in the lower right and then click the Browse button in the resulting dialog box. Now browse and select the new INF file, click Open, and then click OK. You will now be able to see one or more matching driver configurations.

If the INF file contains more than one configuration that matches the device’s Plug and Play ID then multiple selections will appear. Multiple selections may also appear if any other INF file in the folder that you just selected contains a matching Plug and Play ID. You may also see selections for other drivers in the Driver Store that match the ID. Select the desired driver package from the list, then click the Next button, and the driver installation process will begin.

To uninstall a driver, right-click the device in Device Manager and select Uninstall, then click OK to uninstall. Check the “Delete the driver software for this device” checkbox to additionally remove the driver package from the Driver Store. This is equivalent to using PnPUtil with the `-d` argument to remove the driver from the Driver Store.

How to Install/Remove Drivers Using Sound Control Panel

You can accomplish the same functionality from the Sound dialog box in the Control Panel. In the Playback or Recording tab, double-click a device in the list to bring up its main property page. Then click on the Properties button on the right side of the main property page to bring up an additional property page. Click Change Settings in the bottom left of this page, and then click the Driver tab at the top. You can then click Update Driver, which is equivalent to selecting Update Driver Software from the drop-down menu in Device Manager. All of the remaining steps are the same as the steps for using Device Manager.

Driver Install Frameworks

Windows 7 also provides better support for installing drivers programmatically, rather than using the manual procedures described above. You can use the DPInst.exe application to put a simple wrapper around your INF-based driver package to avoid the manual steps shown above.

If your driver needs to be installed as part of a .MSI Microsoft Installer package, you can use DifXApp to install the drivers as part of the larger installer package. Alternately, you can use the DifX API to programmatically install driver packages. For more information, search Microsoft's MSDN Web site for "Driver Install Frameworks Overview".

Using SetupAPI.Device.Log to Debug Driver Installations

You can find the SetupAPI.Device.Log text file in the C:\Windows\INF directory. This file grows a bit each time that the user attempts to install a new device. Start at the end of the file and scroll backwards to see the most recently installed device.

At the very end of the file, you will find an entry that indicates whether the most recent driver installation succeeded or failed:

```
<<< Section end 2011/06/27 09:04:27.251
<<< [Exit status: SUCCESS]
```

If you scroll up, you will eventually come to a blank line followed by the start of the section describing the device installation. It will start with >>> and look something like this:

```
>>> [Device Install (DiShowUpdateDevice) HDAUDIO\FUNC_01&
        VEN_1013&DEV_4206&SUBSYS_106B1800
>>>  Section start 2011/06/27 08:55:32.220
```

The details of the driver installation are shown between these two sections. If the exit status is `FAILURE`, then look for exclamation marks in the left margin. These are placed in front of any line that doesn't fully succeed. Note that installing unsigned drivers can result in a number of exclamation marks that can be ignored.

If your review of the log file doesn't provide any insights, then try running the ChkINF Perl scripts that are included with the Windows Driver Kit (WDK). You must also have Perl installed on your system in order for ChkINF to work properly. ChkINF parses an INF file and generates an HTML file with a list of warnings and errors that are discovered. The INFTest logo test is a wrapper around the ChkINF Perl scripts that is used to qualify an INF file for logo. Your INF file must be able to pass INFTest in order to get logo certification.

Popular Audio Registry Key Locations

When you install an audio device, a number of different registry locations are set to specific values that define the functionality of the audio device. If you are debugging driver installation issues, you may find it useful to memorize these registry locations using RegEdit's Favorites menu:

- `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\DeviceClasses\{6994AD04-93EF-11D0-A3CC-00A0C9223196}`

  This location is the staging area where all of the possible configurations from the INF file are stored. When a driver is actually installed, then Windows migrates these values to matching hardware endpoints.

- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\MMDevices\Audio`

  This location is used to contain settings that have been migrated from the INF file for both the audio endpoint and the associated APO.

- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\MMDevices\DefaultDeviceHeuristics`

  This location is used to control the heuristics for determining the Default Console Device and the Default Communications Device.

- `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{4D36E96C-E325-11CE-BFC1-08002BE10318}`

  This location is used to store keys specific to the device driver (not the endpoint), such as PinConfigOverrideSettings and PowerSettings.

- `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\HDAUDIO`

  This location contains a list of the currently installed Plug and Play IDs for HD Audio devices.

- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Audio`

  This location is used to store global control of audio, such as the flag to allow unsigned APOs to be run for development purposes.

Chapter 13

Windows® Logo Testing for HD Audio

Sometimes when you innovate, you make mistakes. It is best to admit them quickly and get on with improving your other innovations.

—Steve Jobs

Manufacturers of audio hardware such as laptop and desktop computers or USB-connected sound cards and headphones often need to obtain Microsoft Windows Logo Certification for their product to be viable in the marketplace. The process consists of running a series tests on the hardware defined by Microsoft to ensure that the hardware meets a defined audio quality standard and compatibility with Windows requirements. When the tests successfully pass and Microsoft certifies the results, the OEM or system manufacturer can then display the "Windows®7" logo, shown in Figure 13.1, on the system packaging. Externally connected devices, such as a USB headset, can qualify to display "Compatible with Windows®7" by passing the device tests. Complete systems can pass the system tests only if all of the devices on the system have logoed driver packages..

Microsoft Windows Hardware Quality Lab (WHQL) Logo Process

This testing process is managed by the Driver Test Manager (DTM), a testing environment consisting of a controller and an audio test system with software. This and the support hardware are configured as an isolated and private network with the System Under Test. The package typically includes a Domain Controller and may be as simple as a single test station or in an enterprise environment could be hundreds of independent test stations. The DTM system is used for Windows driver testing and consists of hundreds of individual tests required to test specific hardware such as video cards, network adapters, hard drive controllers, and all of the hardware devices found in a PC. Audio hardware is unique in that it requires special tests for quality verification that can only be performed using sophisticated audio test hardware. These are called *audio fidelity tests* and consist of a suite of audio tests to characterize the audio performance of the audio hardware against an established set of specifications.

Figure 13.1 System and Device logos

The WLK Includes the Driver Test Manager (DTM)

DTM is an application and collection of tools that manage tests for development teams who create Windows hardware, drivers, and validate Windows configurations. DTM provides support for automating Windows-based driver and hardware testing and is directed at those who use the WLK to test, validate, and certify drivers.

The DTM system runs Microsoft software called Windows Logo Kit (WLK) that contains all the tests required. The current WLK version at this time of writing is WLK 1.6. After setup, the testing process is mostly automated although certain tests require operator intervention to connect cables to various analog endpoints such as Line Out, Line In, and Microphone In, which use analog 3.5mm jacks.

Windows Quality Online Services (WinQual)

All logo submissions are performed using Microsoft's WinQual Web site, at https://winqual.microsoft.com. In order to join WinQual and gain access to the secure portion of the site, your company must obtain a VeriSign Microsoft Authenticode Code Signing Digital Certificate. This certificate typically costs USD 499 (US dollars) per year, and must be kept current to retain access to the secure portions of the WinQual site. There is a link on the WinQual Web site to acquire this certificate. There is also a link on the WinQual home page to a list of legal documents that your company must sign before you can submit a product for logo. WinQual also includes LogoPoint, which is the repository of logo requirements, and Windows Error Reporting (WER), which keeps track of which devices are crashing the most often. Your company must agree to participate in WER in order to make logo submissions, and you will be notified if your products are causing crashes in the field.

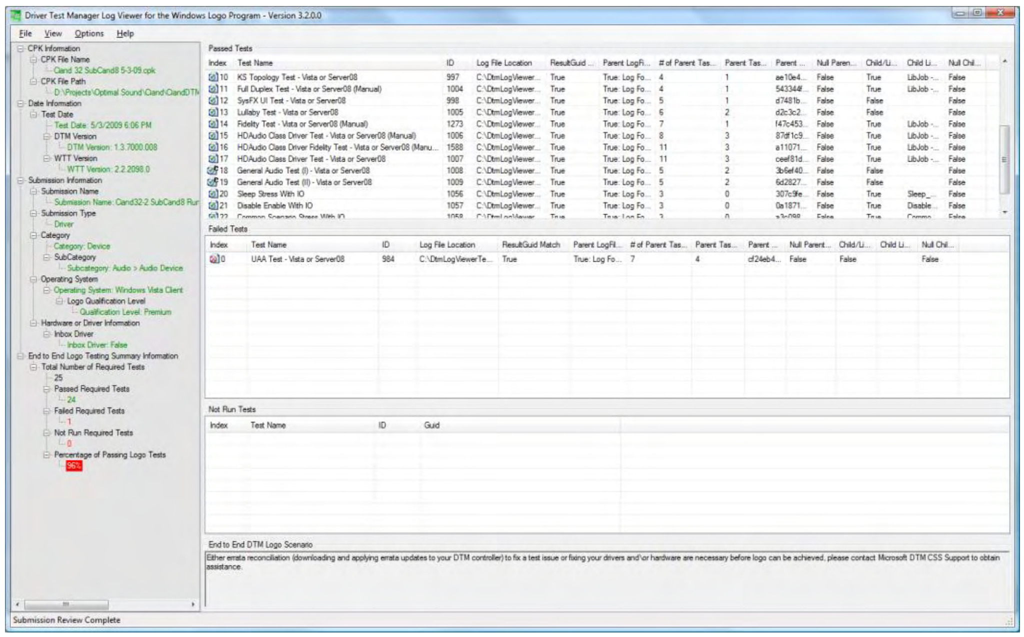
The remainder of this chapter assumes that you have full access to the WinQual Web site. Otherwise you may not be able to perform some of the steps described here.

Driver Submission Process—What Files Are Submitted

To obtain a Windows Logo Certification, you must submit passing log files as well as a copy of all the driver files (DLLs, SYSs, APOs, and INFs). Microsoft defines a specific process for bundling these files together in a "cabinet" or .cab file.

The passing DTM log file consists of a single .CPK file for each operating system. Typically this includes at least one 64-bit OS and optionally at least one 32-bit OS for each driver package. The .CPK file is a compiled binary file that bundles together the test results for all of the tests run as part of the DTM test suite and includes detailed information on the driver, the test system, and the test environment. In addition to several driver reliability and other tests, this includes the audio fidelity tests using both the Class Driver and the Vendor Driver (if present). Every test must pass; even a single failure of one test among the hundreds of required tests will cause a failure of the submission process. There are occasionally some exceptions where certain test failures may be overlooked and Windows certification granted manually but this involves discussion with Microsoft and is usually only granted if the failure can be shown to have been caused by an error of the DTM system, not a real failure of the driver.

The .CPK file may be viewed using the DTM Log Viewer, which will generate an explorer view of all of the tests and allow expansion of these results, as shown in Figure 13.2.



Driver Test Manager Log Viewer for the Windows Logo Program - Version 3.2.0.0

```
CPK Information
    CPK File Name
        Cand 32 SubCand8 5-3-09.cpk
    CPK File Path
        D:\Projects\Optimal Sound\Cand\CandDTM
Date Information
    Test Date
        Test Date: 5/3/2009 6:06 PM
        DTM Version
            DTM Version: 1.3.7000.008
        WTT Version
            WTT Version: 2.2.2098.0
Submission Information
    Submission Name
        Submission Name: Cand32-2 SubCand8 Run
    Submission Type
        Driver
    Category
        Category: Device
        SubCategory
            Subcategory: Audio > Audio Device
    Operating System
        Operating System: Windows Vista Client
            Logo Qualification Level
                Qualification Level: Premium
    Hardware or Driver Information
        Inbox Driver
            Inbox Driver: False
End to End Logo Testing Summary Information
    Total Number of Required Tests
        25
        Passed Required Tests
            24
        Failed Required Tests
            1
        Not Run Required Tests
            0
        Percentage of Passing Logo Tests
            96%
```

Passed Tests

Index	Test Name	ID	Log File Location	ResultGuid ...	Parent LogFi...	# of Parent Tas...	Parent Tas...	Parent ...	Null Paren...	Child/Li...	Child Li...	Null Chil...
10	KS Topology Test - Vista or Server08	997	C:\DtmLogViewer...	True	True: Log Fo...	4	1	ae10e4...	False	True	LibJob -...	False
11	Full Duplex Test - Vista or Server08 (Manual)	1004	C:\DtmLogViewer...	True	True: Log Fo...	4	1	543344f...	False	True	LibJob -...	False
12	SysFX UI Test - Vista or Server08	998	C:\DtmLogViewer...	True	True: Log Fo...	5	1	d7481b...	False	False		False
13	Lullaby Test - Vista or Server08	1005	C:\DtmLogViewer...	True	True: Log Fo...	6	2	d2c3c2...	False	False		False
14	Fidelity Test - Vista or Server08 (Manual)	1273	C:\DtmLogViewer...	True	True: Log Fo...	7	1	f47c453...	False	True	LibJob -...	False
15	HDAudio Class Driver Test - Vista or Server08 (Manual)	1006	C:\DtmLogViewer...	True	True: Log Fo...	8	3	87df1c9...	False	True	LibJob -...	False
16	HDAudio Class Driver Fidelity Test - Vista or Server08 (Manu...	1588	C:\DtmLogViewer...	True	True: Log Fo...	11	3	a11071...	False	True	LibJob -...	False
17	HDAudio Class Driver Test - Vista or Server08	1007	C:\DtmLogViewer...	True	True: Log Fo...	11	3	ceef81d...	False	True	LibJob -...	False
18	General Audio Test (I) - Vista or Server08	1008	C:\DtmLogViewer...	True	True: Log Fo...	5	2	3b6ef40...	False	False		False
19	General Audio Test (II) - Vista or Server08	1009	C:\DtmLogViewer...	True	True: Log Fo...	5	2	6d2827...	False	False		False
20	Sleep Stress With IO	1056	C:\DtmLogViewer...	True	True: Log Fo...	3	0	307c9fe...	False	True	Sleep_...	False
21	Disable Enable With IO	1057	C:\DtmLogViewer...	True	True: Log Fo...	3	0	0a1871...	False	True	Disable...	False
22	Common Scenario Stress With IO	1058	C:\DtmLogViewer...	True	True: Log Fo...	3	0	a3c098...	False	True	Commo...	False

Failed Tests

Index	Test Name	ID	Log File Location	ResultGuid Match	Parent LogFi...	# of Parent Tas...	Parent Tas...	Parent ...	Null Parent...	Child/Li...	Child Li...	Null Chil...
0	UAA Test - Vista or Server08	984	C:\DtmLogViewerTe...	True	True: Log Fo...	7	4	cf24eb4...	False	False		False

Not Run Tests

Index	Test Name	ID	Guid

End to End DTM Logo Scenario

Either errata reconciliation (downloading and applying errata updates to your DTM controller) to fix a test issue or fixing your drivers and\or hardware are necessary before logo can be achieved, please contact Microsoft DTM CSS Support to obtain assistance.

Submission Review Complete

Figure 13.2 DTM Log Viewer shows the contents of a .CPK file.

WLK Errata

When Microsoft discovers faults or tests that may cause unintended failures in the DTM test software, they generate what are called *QFE*s (Quick Fixes) that can be downloaded and added to an existing DTM test environment. Installing QFEs on a DTM system will often clear tests that have failed, changing them to passed results. For this reason, it is important to keep a DTM system up to date with the latest QFEs to avoid finding errors that are due to DTM test system faults. QFEs are downloaded from the WinQual Web site and easily installed into the DTM system.

Third-Party Audio Fidelity Testing

Since the audio fidelity testing portion of the DTM test process requires both specialized equipment and expert audio testing knowledge, Microsoft has included a mechanism to allow third-party test houses with this equipment and knowledge to perform the necessary audio fidelity tests and generate test logs that the driver developer can then import into their DTM test environment to combine with the regular driver tests to complete the submission process. The driver developer will run all the necessary tests on a target device or system omitting only the audio fidelity tests. The third party test facility will run just the audio fidelity tests on an identical device or system and then the two sets of results can be merged to create a complete .CPK file ready for submission. It is important that the device or systems be identical and configured identically so the results will match. If not, it will not be possible to import the audio fidelity test logs into the primary DTM test environment. Microsoft provides a list of third-party test facilities that can provide this service on their Web site. Just search for “audio fidelity testing”.

Test Signing

The Windows 64-bit operating system requires that all kernel-mode drivers be *signed*. This presents an interesting catch-22 situation since the drivers being tested are not yet signed, yet they must pass a “signability” test in order to be installed and then subsequently signed. Microsoft provides for this with a process called *Test Signing*. This procedure test signs the drivers in such a way that a 64-bit installation will load test-signed drivers. Note that you can’t distrbute drivers that have been test signed; this is only for internal testing. The process is somewhat complex but if you carefully follow the step-by-step instructions available on the Microsoft WinQual site, you can obtain test-signed versions of your drivers to proceed with the DTM testing. You must have a WinQual account and a Verisign certificate to test sign your drivers, and your company must execute the test-signing legal agreement.

Audio Fidelity Requirements

Microsoft takes an occasional snapshot of the audio fidelity requirements from time to time, and makes this information publicly available. However, a WinQual account is required to view current requirements.

In-Box Class Drivers

Microsoft includes with Windows a set of Class Drivers for all UAA-compatible audio hardware. These generic drivers are designed to support all of the normal features of a device. Hardware manufacturers typically also develop device-specific drivers with extended capabilities compared to the generic class driver. It is also possible to extend the Class Driver functionality to create a driver package using filter drivers and APOs along with the Class Driver. DTM requires and tests for compliance with both the class drivers and the third-party drivers. A set of HD Audio Class Driver Tests will re-run various audio device tests to test the compliance these devices with the Microsoft HD Audio Class Driver. All audio tests must pass with both the Class Driver and any Third Party driver package, if present.

Test Duration

The time required to run a complete set of Audio Fidelity tests on a 32- or 64-bit system or driver will vary depending on how many analog end points are in the system. With a typical Mic In, Line In, and Headset or Speaker Out, a full test run would typically be about an hour. The test requires occasional operator action: for example, answering questions such as "Is the Speakers end point actually a Line Out end point?" The test also requires the operator to insert 3.5mm plugs into specific analog end points such as "connect the Audio Precision Generator Output to the Mic input". Systems that are tested for both 64- and 32-bit operating systems that include front panel microphone and headset jacks and rear panel microphone, line-in, and speaker-out jacks will typically take two to three hours for a complete test run. If you make an error plugging in a cable, for example by plugging the generator into the front panel microphone jack when it wanted the rear panel microphone jack, the test will fail on the first level setting test and the entire process will have to be repeated.

DTM System

DTM is comprised of three installation components, the Controller, the Studio, and the Client.

DTM Controller. The DTM Controller manages tests that are run on available clients. The DTM Controller is installed from the DTM installation media and contains separate installers to install the DTM Studio and DTM Client components. You must install DTM Controller first from the DTM installation media. You can then install DTM Studio and DTM Client.

DTM Controllers are computers that control what tests run on DTM clients and when those tests run. Controllers host a SQL database that stores tests to be run and the results of previous tests for review and future use. The DTM installation will install a limited version of SQL Server that has a maximum storage capacity of 2 GB. If you will be testing a large number of systems, you should upgrade to the full version of SQL Server so that you don't run out of storage space.

DTM Studio. You use DTM Studio to select and schedule tests to be run on available clients that are connected to, and controlled by, the DTM Controller computer. You will not see an option to install the DTM Studio from the DTM installation media. Instead you install DTM Studio directly from a shared network location on the DTM Controller. For smaller DTM setups, you will typically run DTM Studio on the same PC as the DTM controller, but it is possible to run it on a separate system as well.

DTM Studio is the application and user interface that testers use to create and schedule tests and otherwise manipulate controllers and clients. With DTM Studio, you can organize the computers in your lab into the appropriate environment to test your drivers.

DTM Client. Each client computer can have a different configuration, appropriate for your testing scenario(s) including different hardware, operating systems, service packs, drivers, and so forth. You will not see an option to install the DTM Client from the DTM installation media. Instead, you install DTM Client directly on your individual test computers from a shared network location on the DTM Controller.

DTM Clients are computers that run tests. You should have as many client computers with as many different configurations as possible so that your driver can be tested under the widest variety of real-world conditions. Clients should have different hardware and software configurations and different operating systems and service packs installed.

A DTM Test System for Audio Driver testing requires three computers and an audio test instrument, which must be an *Audio Precision System Two 2700*. The 2500 series will also work in most cases. System Two consists of a dual channel low-distortion, high-quality reference signal generator, an accurate high-quality signal level meter, a high-quality total harmonic distortion meter, a sensitive noise meter, frequency counter, and a phase meter. Additionally, System Two provides several measurement conditioning capabilities that allow it to measure to specific international standards. For example, noise is typically measured with a standard-specified bandwidth and commonly with a standard-specified frequency weighting such as ANSI-IEC A-weighting. Certain tests require these bandwidth conditioning settings and System Two provides them. If you don't have these add-ons installed in the Audio Precision, then the tests will usually run, but may not provide passing results until the add-on filters are installed.

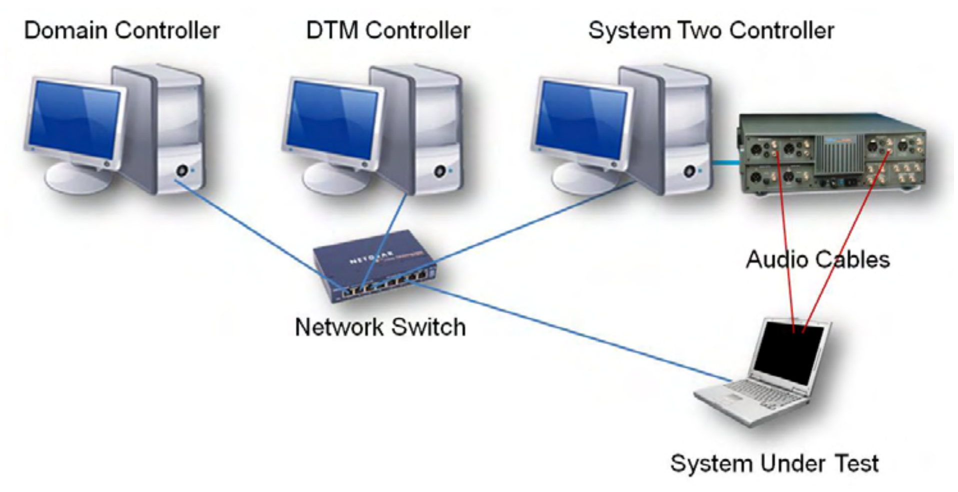


Figure 13.3 DTM Test System components

Components of a DTM System

A DTM system typically consists of a Domain Controller, a DTM Controller, and Audio Test Controller, an Audio Precision System Two test set, and the System Under Test. Each of these computers is connected in a network. This network must be isolated and not be part of a corporate network.

DTM Domain Controller

The DTM Domain Controller runs Windows Server (2003 or 2008). It also runs:

File Server

DNS Server

DHCP Server

The Domain Controller provides an independent local domain and network not connected to the corporate network. All firewall and antivirus protection on all systems should be disabled. It is possible to configure DTM to work on a domain within a workgroup. The Domain Controller is needed only for the domain scenario. If present, the Domain Controller must be installed on a separate instance of Windows Server, and cannot be combined with the DTM Controller. Virtualization is not supported by DTM, so this will require a physically separate server.

Also note that this network should be an isolated network, and *not* connected to your corporate network or to the Internet. This is because the *System(s) Under Test* should be configured without firewall or virus checker, and set to auto-login. This leaves the test network open to virus threats if connected to the Internet.

DTM Controller

The PC used to host the DTM Controller must run Windows Server 2008 R2 64-bit. It has hardware requirements of 2 GHz Dual-Core CPU with 2 GB RAM or more. This controller runs Microsoft WLK 1.6, the test software containing all of the individual tests. The user interface for the DTM system is DTM Studio, and this software may be run on any computer on this network but it is convenient to run it on this DTM Controller PC.

Audio Precision Host

The Audio Precision Host runs Audio Precision APWIN software AP2700 v 3.30, the software that controls the System Two. You can download this software from www.audioprecision.com. This software is in turn controlled by the DTM software WLK. The operating system on this PC may be Windows Vista or Windows 7. This system also runs DTM Client, the program that connects this PC to the DTM environment. This PC is joined to the local test domain.

You must create an Audio Precision Host system and install the APWIN software regardless of whether you do or don't have an Audio Precision test set. This is necessary to import fidelity test results from a third-party test house in the case where you do not have an Audio Precision test set.

Audio Precision System Two

Audio Precision manufactures several models of the audio test instrument System Two. The models that will function in the DTM system include: SYS-2702, SYS-2712, SYS-2522, and SYS-2722. In addition, the System Two must be equipped with certain specific options to be able to perform the tests properly. The options include:

- *A-Weighting noise filter. This provides frequency weighting required for certain noise measurement.*
- *AES-17 Filter Package. This filter provides a 20-kHz steep roll off low pass filter and is required to reduce out-of-band noise typically present on oversampling converters.*

What Tests Are Run: Current Audio Fidelity Testing

The Fidelity Test verifies that the audio device meets the WLP requirements for a high-fidelity audio playback experience. The test plays a tone that is analyzed by an Audio Precision System Two analyzer that is connected to the analog output jack of the system. In other cases, the *System Under Test* generates a test tone that is measured by the Audio Precision. The following test cases appear in the DTM for both Basic and Premium logos. A set of HD Audio Class Driver Tests will re-run various audio device tests to test the compliance of these devices with the Microsoft HD Audio Class Driver. Fidelity Test takes eight render measurements on analog outputs:

- *Output level.* This test plays a digital full-scale signal and verifies that it meets the Full-Scale Output Voltage requirement in the Windows Logo Program Device Fidelity Requirements.
- *Dynamic range.* This test takes a noise floor measurement in the presence of a signal per AES-17 and verifies that the measurement meets the Dynamic Range requirement in the Windows Logo Program Device Fidelity Requirements.

- *THD+N.* This test takes a THD+N measurement and verifies the measurement per the Windows Logo Program Device Fidelity Requirements document.
- *Magnitude response.* This test measures the frequency response of the device and verifies whether it meets the Magnitude Response requirement in the Windows Logo Program Device Fidelity Requirements.
- *Phase Delay.* This test measures the interchannel phase delay and verifies that it meets the requirements.
- *System Activity.* This test measures the noise level of the system during system activity (GPU activity, disk activity, and so on) and verifies that it meets the requirements.
- *Skew.* This test plays a high-frequency 15 kHz tone with strong settling parameters and verifies via the analyzer that the fundamental frequency is within the frequency requirements of the expected value.
- *Power State Transition.* This test measures the noise level of the system during transition to low power and back to normal audio playback. This measurement takes place while power is removed and reapplied to the HD Audio codec. If any pops and clicks exceed the specified level, then the test will fail.

Fidelity Test also takes five capture measurements on analog inputs:

- *Input level.* This test plays analog signals of varying voltages into the recording jack and attempts to find the voltage that induces a full-scale signal (or equivalent, per AES6id) in the digital domain. It does this by gradually increasing the input level until it just measures a THD+N value of 1 percent (-40dB) and sets this level as the full scale level. Be careful to ensure that your system is designed to handle the levels coming from the Audio Precision, to prevent codec latchup.
- *Skew.* This test captures a large sample of a high-frequency 15 kHz tone generated from the analyzer and verifies via DSP that the fundamental frequency is within the requirements of the expected value.
- *THD+N.* This test captures a signal and measures the THD+N via DSP. It then validates that the harmonics and noise level is lower than the requirement allows.

- *Dynamic range.* This test captures a signal and takes a noise measurement, and verifies that the dynamic range is at least as high as required.
- *Frequency response.* This test captures signals at various frequency levels and verifies that the measured intensity of the signal across the audio frequency range does not vary more than the requirement allows.

All test cases return pass or fail, and the generated log file contains the actual measurement that is recorded.

Fidelity Test requires special hardware to run. The test requires the presence of a System Two series or an AP2700 series audio analyzer/generator from Audio Precision.

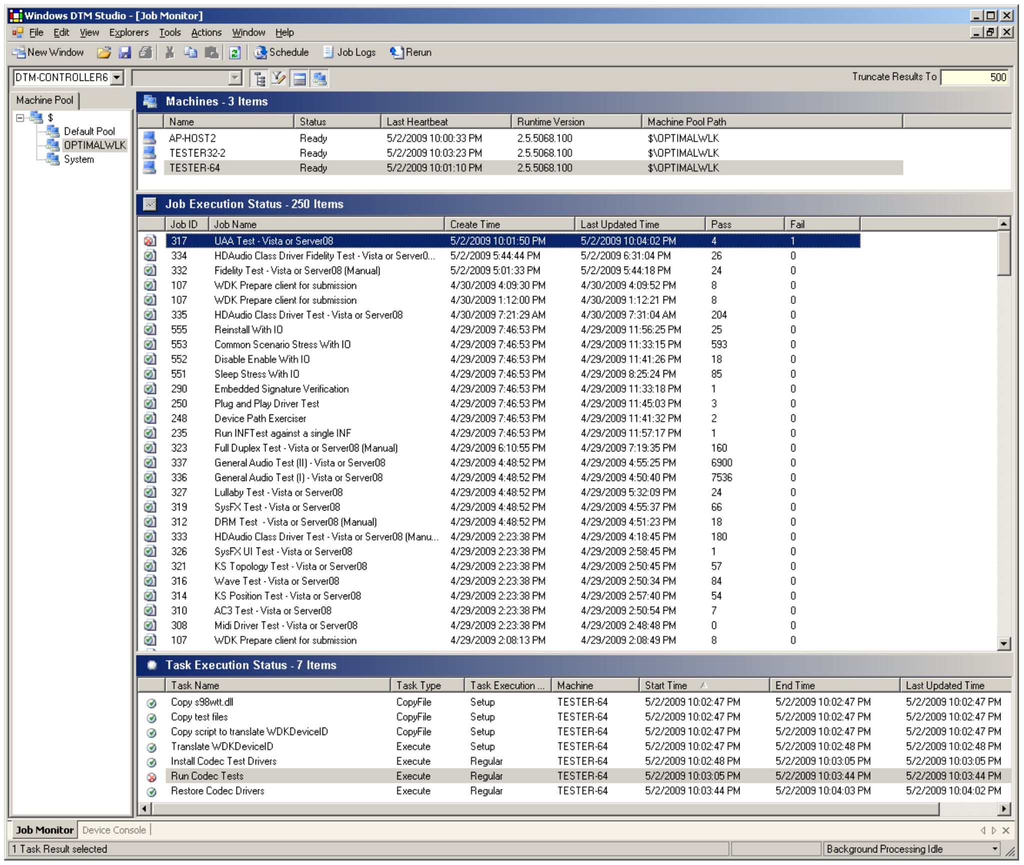
Fidelity Test will test only:

- Analog line out jacks
- Analog headphone out jacks
- Analog speaker jacks (which are usually actually Line Out jacks)
- Analog line in jacks
- Analog microphone input jacks

Integrated speakers and integrated microphones are not tested; neither are digital connectors such as S/PDIF or HDMI.

Starting DTM Controller

On the DTM Controller computer, launch Studio. Initially, the Start Page will open. Go to Explorers, Job Monitor. This will open an explorer with a Machine Pool pane on the left, Machines pane, Job Status pane, and Task Status panes on the right, as shown in Figure 13.4.



Windows DTM Studio - [Job Monitor]

DTM-CONTROLLER6 — Truncate Results To 500

Machine Pool: $ — Default Pool, OPTIMALWLK, System

Machines - 3 Items

Name	Status	Last Heartbeat	Runtime Version	Machine Pool Path
AP-HOST2	Ready	5/2/2009 10:00:33 PM	2.5.5068.100	$\OPTIMALWLK
TESTER32-2	Ready	5/2/2009 10:03:23 PM	2.5.5068.100	$\OPTIMALWLK
TESTER-64	Ready	5/2/2009 10:01:10 PM	2.5.5068.100	$\OPTIMALWLK

Job Execution Status - 250 Items

Job ID	Job Name	Create Time	Last Updated Time	Pass	Fail
317	UAA Test - Vista or Server08	5/2/2009 10:01:50 PM	5/2/2009 10:04:02 PM	4	1
334	HDAudio Class Driver Fidelity Test - Vista or Server0...	5/2/2009 5:44:44 PM	5/2/2009 6:31:04 PM	26	0
332	Fidelity Test - Vista or Server08 (Manual)	5/2/2009 5:01:33 PM	5/2/2009 5:44:18 PM	24	0
107	WDK Prepare client for submission	4/30/2009 4:09:30 PM	4/30/2009 4:09:52 PM	8	0
107	WDK Prepare client for submission	4/30/2009 1:12:00 PM	4/30/2009 1:12:21 PM	8	0
335	HDAudio Class Driver Test - Vista or Server08	4/30/2009 7:21:29 AM	4/30/2009 7:31:04 AM	204	0
555	Reinstall With IO	4/29/2009 7:46:53 PM	4/29/2009 11:56:25 PM	25	0
553	Common Scenario Stress With IO	4/29/2009 7:46:53 PM	4/29/2009 11:33:15 PM	593	0
552	Disable Enable With IO	4/29/2009 7:46:53 PM	4/29/2009 11:41:26 PM	18	0
551	Sleep Stress With IO	4/29/2009 7:46:53 PM	4/29/2009 8:25:24 PM	85	0
290	Embedded Signature Verification	4/29/2009 7:46:53 PM	4/29/2009 11:33:18 PM	1	0
250	Plug and Play Driver Test	4/29/2009 7:46:53 PM	4/29/2009 11:45:03 PM	3	0
248	Device Path Exerciser	4/29/2009 7:46:53 PM	4/29/2009 11:41:32 PM	2	0
235	Run INFTest against a single INF	4/29/2009 7:46:53 PM	4/29/2009 11:57:17 PM	1	0
323	Full Duplex Test - Vista or Server08 (Manual)	4/29/2009 6:10:55 PM	4/29/2009 7:19:35 PM	160	0
337	General Audio Test (II) - Vista or Server08	4/29/2009 4:48:52 PM	4/29/2009 4:55:25 PM	6900	0
336	General Audio Test (I) - Vista or Server08	4/29/2009 4:48:52 PM	4/29/2009 4:50:40 PM	7536	0
327	Lullaby Test - Vista or Server08	4/29/2009 4:48:52 PM	4/29/2009 5:32:09 PM	24	0
319	SysFX Test - Vista or Server08	4/29/2009 4:48:52 PM	4/29/2009 4:55:37 PM	66	0
312	DRM Test - Vista or Server08 (Manual)	4/29/2009 4:48:52 PM	4/29/2009 4:51:23 PM	18	0
333	HDAudio Class Driver Test - Vista or Server08 (Manu...	4/29/2009 2:23:38 PM	4/29/2009 4:18:45 PM	180	0
326	SysFX UI Test - Vista or Server08	4/29/2009 2:23:38 PM	4/29/2009 2:58:45 PM	1	0
321	KS Topology Test - Vista or Server08	4/29/2009 2:23:38 PM	4/29/2009 2:50:45 PM	57	0
316	Wave Test - Vista or Server08	4/29/2009 2:23:38 PM	4/29/2009 2:50:34 PM	84	0
314	KS Position Test - Vista or Server08	4/29/2009 2:23:38 PM	4/29/2009 2:57:40 PM	54	0
310	AC3 Test - Vista or Server08	4/29/2009 2:23:38 PM	4/29/2009 2:50:54 PM	7	0
308	Midi Driver Test - Vista or Server08	4/29/2009 2:23:38 PM	4/29/2009 2:48:48 PM	0	0
107	WDK Prepare client for submission	4/29/2009 2:08:13 PM	4/29/2009 2:08:49 PM	8	0

Task Execution Status - 7 Items

Task Name	Task Type	Task Execution ...	Machine	Start Time	End Time	Last Updated Time
Copy s98wtt.dll	CopyFile	Setup	TESTER-64	5/2/2009 10:02:47 PM	5/2/2009 10:02:47 PM	5/2/2009 10:02:47 PM
Copy test files	CopyFile	Setup	TESTER-64	5/2/2009 10:02:47 PM	5/2/2009 10:02:47 PM	5/2/2009 10:02:47 PM
Copy script to translate WDKDeviceID	CopyFile	Setup	TESTER-64	5/2/2009 10:02:47 PM	5/2/2009 10:02:47 PM	5/2/2009 10:02:47 PM
Translate WDKDeviceID	Execute	Setup	TESTER-64	5/2/2009 10:02:47 PM	5/2/2009 10:02:48 PM	5/2/2009 10:02:48 PM
Install Codec Test Drivers	Execute	Regular	TESTER-64	5/2/2009 10:02:48 PM	5/2/2009 10:03:05 PM	5/2/2009 10:03:05 PM
Run Codec Tests	Execute	Regular	TESTER-64	5/2/2009 10:03:05 PM	5/2/2009 10:03:44 PM	5/2/2009 10:03:44 PM
Restore Codec Drivers	Execute	Regular	TESTER-64	5/2/2009 10:03:44 PM	5/2/2009 10:04:03 PM	5/2/2009 10:04:02 PM

1 Task Result selected — Background Processing Idle

Figure 13.4 DTM Job Monitor in DTM Studio is the user interface to control the DTM activity.

DTM Studio Job Monitor

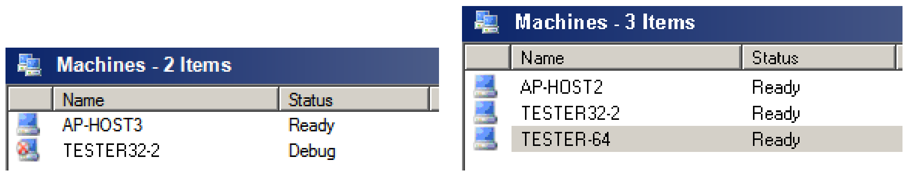
The DTM Job Monitor is the primary user interface to observe system activity and progress. It lists all machines (PCs) in the pool and when tests are running, shows each of the jobs and progress. When a job (or test) completes, it shows whether the test passed or failed and lists all the tests that have been run.

Creating a Machine Pool

When DTM is started initially, there will be no Machine Pools, only a Default pool. Create a new machine pool to host the computers that will run the tests and be tested. Right-click on the "$" symbol under Machine Pools and select Create New Pool, giving it an appropriate name.

Adding systems to the pool

With the new machine pool set up, you can move computers from the default pool to the new machine pool. Just right click and drag the computers to the new pool. Once in the pool, set the computer to Ready. To do this, right click on the computer and select Reset. After a minute or so, the computer status should change from a red Debug to a blue Ready. Note that Explorer does not self refresh, you need to click the refresh button at the top menu bar or hit F5.



Machines - 2 Items

Name	Status
AP-HOST3	Ready
TESTER32-2	Debug

Machines - 3 Items

Name	Status
AP-HOST2	Ready
TESTER32-2	Ready
TESTER-64	Ready

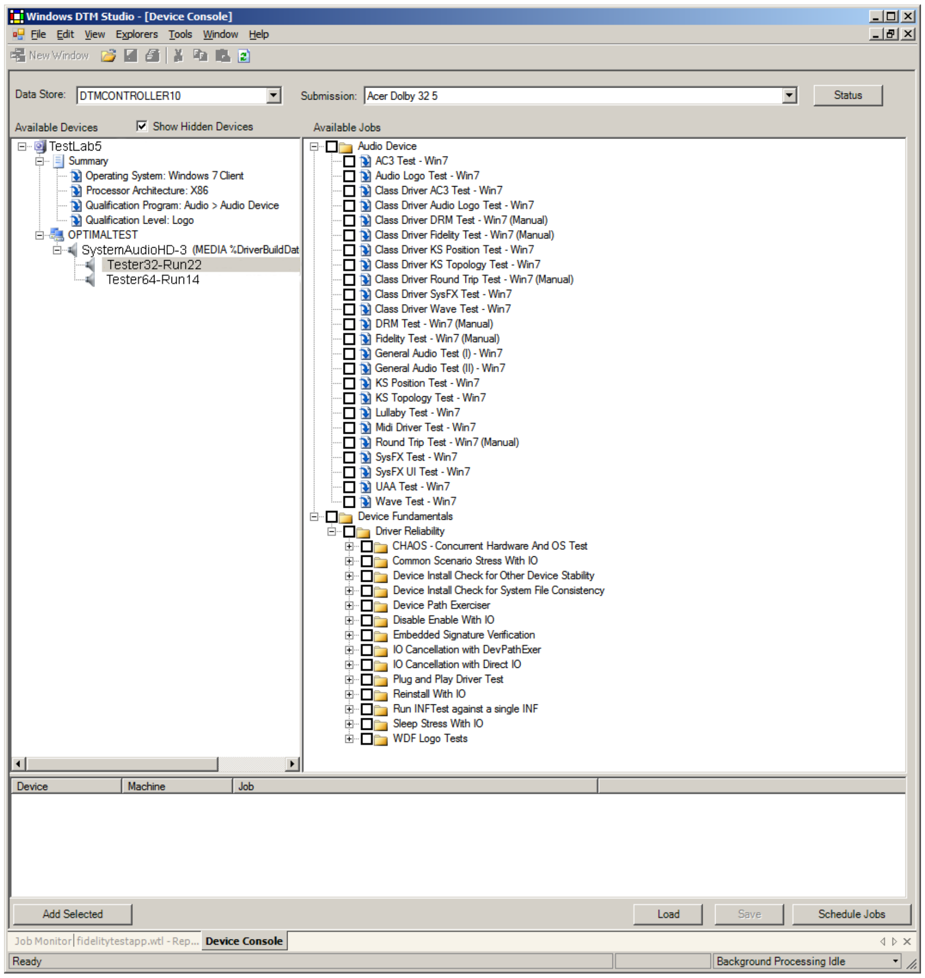
Figure 13.5 Status mode of machines. On the left, one machine is not ready.

What to do when you can't get a heartbeat

The DTM system constantly monitors the pulse of all systems in the pool. If a system should lose connection with the DTM Controller for any reason, the controller will show the Status of that system as other than Ready and the blue icon will be replaced with a red X. The DTM system will report a lack of heartbeat from that system. Unless the system has failed, it should be possible to restore the heartbeat and return the status to Ready. Start by opening the status by right clicking on the status, select Change Status and select Reset and see if the system returns to Ready after a few minutes. If it does not, try selecting "Unsafe", then Reset and see if it returns to Ready. If that does not solve the problem, set the system to status Unsafe, remove it from the pool by dragging the subject machine to the Default Pool, then return it to the test pool and set status Reset. If this does not work, repeat the above process but reboot the subject machine while it is in the default pool. In extreme cases, you may need to reinstall the client on the System Under Test.

Choosing Tests to Run

To select tests to run, first be sure all required machines in the current machine pool are running and have a recent heartbeat. Then select Device Console. On the left, you will see a list of available devices and systems. Highlight the device you want to test. Using the Submission drop down box in the upper right, select Create New Submission, give this test run a name and go through the several panels that ask questions about the kind of test you want to run. These include information about the device or system and the level of qualification such as premium or regular. After you complete this process, the DTM gatherer will populate the right pane with the required tests that will need to be run for a submission.



Windows DTM Studio - [Device Console]

File Edit View Explorers Tools Window Help

New Window

Data Store: DTMCONTROLLER10 Submission: Acer Dolby 32 5 Status

Available Devices ☑ Show Hidden Devices

- TestLab5
  - Summary
    - Operating System: Windows 7 Client
    - Processor Architecture: X86
    - Qualification Program: Audio > Audio Device
    - Qualification Level: Logo
  - OPTIMALTEST
    - SystemAudioHD-3 (MEDIA %DriverBuildDat
      - Tester32-Run22
      - Tester64-Run14

Available Jobs

- Audio Device
  - AC3 Test - Win7
  - Audio Logo Test - Win7
  - Class Driver AC3 Test - Win7
  - Class Driver Audio Logo Test - Win7
  - Class Driver DRM Test - Win7 (Manual)
  - Class Driver Fidelity Test - Win7 (Manual)
  - Class Driver KS Position Test - Win7
  - Class Driver KS Topology Test - Win7
  - Class Driver Round Trip Test - Win7 (Manual)
  - Class Driver SysFX Test - Win7
  - Class Driver Wave Test - Win7
  - DRM Test - Win7 (Manual)
  - Fidelity Test - Win7 (Manual)
  - General Audio Test (I) - Win7
  - General Audio Test (II) - Win7
  - KS Position Test - Win7
  - KS Topology Test - Win7
  - Lullaby Test - Win7
  - Midi Driver Test - Win7
  - Round Trip Test - Win7 (Manual)
  - SysFX Test - Win7
  - SysFX UI Test - Win7
  - UAA Test - Win7
  - Wave Test - Win7
- Device Fundamentals
  - Driver Reliability
    - CHAOS - Concurrent Hardware And OS Test
    - Common Scenario Stress With IO
    - Device Install Check for Other Device Stability
    - Device Install Check for System File Consistency
    - Device Path Exerciser
    - Disable Enable With IO
    - Embedded Signature Verification
    - IO Cancellation with DevPathExer
    - IO Cancellation with Direct IO
    - Plug and Play Driver Test
    - Reinstall With IO
    - Run INFTest against a single INF
    - Sleep Stress With IO
    - WDF Logo Tests

Device	Machine	Job

Add Selected Load Save Schedule Jobs

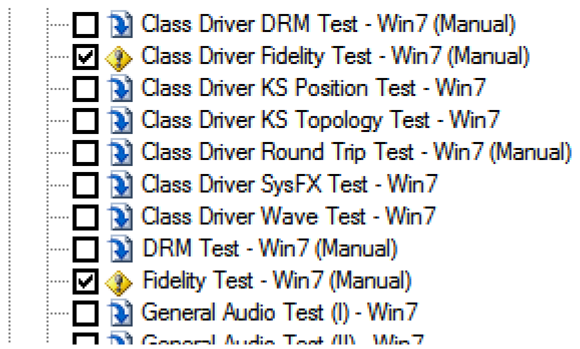
Job Monitor | fidelitytestapp.wtl - Rep... | Device Console

Ready Background Processing Idle

Figure 13.6 Device Console shows a list of available tests and lets you choose which ones to run.

Running Tests

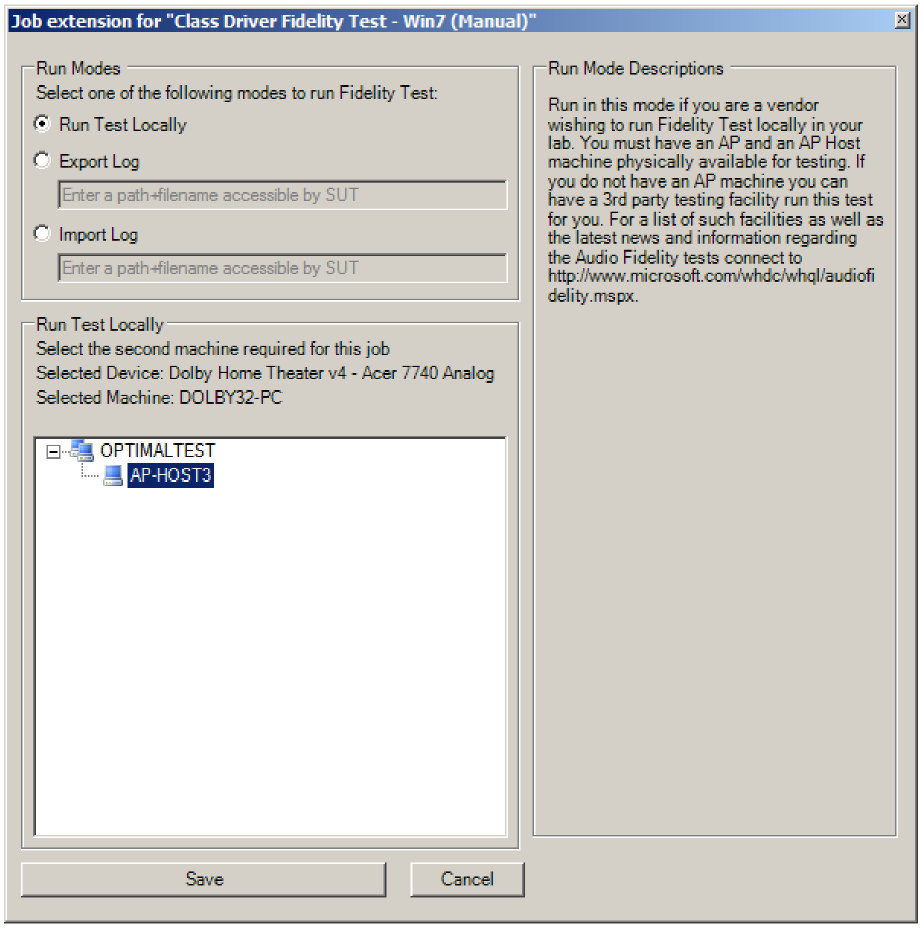
To run a suite of tests, simply check the boxes to the right of the listed tests. All tests on the list with (Manual) next to them require operator action to answer a question to connect a cable. If you select either of the Audio Fidelity tests, you will initially see a yellow indicator:



- ☐ Class Driver DRM Test - Win7 (Manual)
- ☑ Class Driver Fidelity Test - Win7 (Manual)
- ☐ Class Driver KS Position Test - Win7
- ☐ Class Driver KS Topology Test - Win7
- ☐ Class Driver Round Trip Test - Win7 (Manual)
- ☐ Class Driver SysFX Test - Win7
- ☐ Class Driver Wave Test - Win7
- ☐ DRM Test - Win7 (Manual)
- ☑ Fidelity Test - Win7 (Manual)
- ☐ General Audio Test (I) - Win7

Figure 13.7 Fidelity Tests selected showing yellow indicator indicating additional information is required.

This indicates that additional information is required before these tests can be run. Right click on each of these to bring up an additional dialog box.


Job extension for "Class Driver Fidelity Test - Win7 (Manual)"

Run Modes

Select one of the following modes to run Fidelity Test:

[✓ Run Test Locally]

Export Log

Enter a path+filename accessible by SUT

Import Log

Enter a path+filename accessible by SUT

Run Test Locally

Select the second machine required for this job

Selected Device: Dolby Home Theater v4 - Acer 7740 Analog

Selected Machine: DOLBY32-PC

OPTIMALTEST

AP-HOST3

Run Mode Descriptions

Run in this mode if you are a vendor wishing to run Fidelity Test locally in your lab. You must have an AP and an AP Host machine physically available for testing. If you do not have an AP machine you can have a 3rd party testing facility run this test for you. For a list of such facilities as well as the latest news and information regarding the Audio Fidelity tests connect to http://www.microsoft.com/whdc/whql/audiofidelity.mspx.

Figure 13.8 Dialog that opens after right-clicking on a yellow-indicated test.

It is necessary to select the machine that hosts the Audio Precision test software and that controls the Audio Precision System Two. Highlight the machine and click Save.

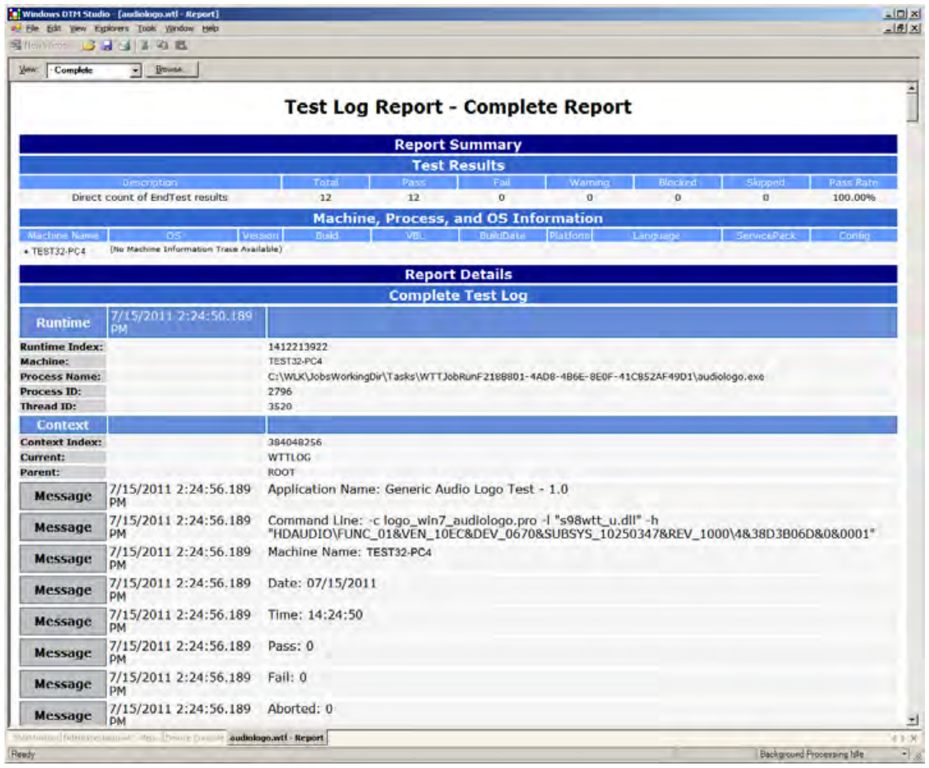
When this is resolved, you can click on Add Selected which will copy all checked tests to the lower pane ready to run. Then click on Schedule Jobs which will add these test jobs top the job queue. You can then return to the job monitor screen, click on Refresh and you should soon see the selected machines status change from Ready to Running.

It is not necessary to run all tests at one time. You can select any subset of tests, run these, and then return to the Device Console to select additional tests to run. If a test fails, you can rerun that test. Ultimately, you need to have run all of the tests in the Device Console list and you need to have 100% PASS for all tests. Even a single failure out of the hundreds of individual tests will fail the submission. We will discuss Audio Fidelity failures below.

Viewing errors, using DTM console or Log Viewer

After running each test, the Job Monitor explorer will identify the passes and failures for every test. See the Job Monitor screen capture above, Fig XX, for an example. You can drill down to more detail for any test to get more information on what failed if you need to. Highlight the test in the middle Job Execution Status pane to see a list of the tasks in the lower Task Execution Status pane. Then right click on the failed task and select either View Task Log or Child Task to expand the tasks to the sub tasks involved. Then right click on the failed Run Test task and select View Task Log. This will open a Test Log Report that is a detailed list of all the activity that was part of the test run, the conditions of the system under test, test activity, and all messages. If there is a failure, it will clearly show in red with the test results.

If you don't have access to the DTM console, you can download DTM Log Viewer from the Microsoft Connect site to view the logs that have been generated. You can save an XML file of this test report by selecting save as and providing a file name for the .WTL file that gets saved. The DTM Log Viewer application will open a .WTL file and show it as it looks here. It is useful to save .WTL files for failed Audio Fidelity tests to assist with seeing what failed and provide guidance with how to solve the failure.



Windows DTM Studio - [audiologo.wtl - Report]
File Edit View Explorers Tools Window Help
View: Complete
Browse...
audiologo.wtl - Report
Ready
Background Processing Idle

Test Log Report - Complete Report

Report Summary

Test Results

Description	Total	Pass	Fail	Warning	Blocked	Skipped	Pass Rate
Direct count of EndTest results	12	12	0	0	0	0	100.00%

Machine, Process, and OS Information

Machine Name	OS	Version	Build	VBL	BuildDate	Platform	Language	ServicePack	Config
• TEST32-PC4	(No Machine Information Trace Available)								

Report Details

Complete Test Log

Runtime	7/15/2011 2:24:50.189 PM	
Runtime Index:		1412213922
Machine:		TEST32-PC4
Process Name:		C:\WLK\JobsWorkingDir\Tasks\WTTJobRunF218B801-4AD8-4B6E-8E0F-41C852AF49D1\audiologo.exe
Process ID:		2796
Thread ID:		3520
Context		
Context Index:		384048256
Current:		WTTLOG
Parent:		ROOT
Message	7/15/2011 2:24:56.189 PM	Application Name: Generic Audio Logo Test - 1.0
Message	7/15/2011 2:24:56.189 PM	Command Line: -c logo_win7_audiologo.pro -l "s98wtt_u.dll" -h "HDAUDIO\FUNC_01&VEN_10EC&DEV_0670&SUBSYS_10250347&REV_1000\4&38D3B06D&0&0001"
Message	7/15/2011 2:24:56.189 PM	Machine Name: TEST32-PC4
Message	7/15/2011 2:24:56.189 PM	Date: 07/15/2011
Message	7/15/2011 2:24:56.189 PM	Time: 14:24:50
Message	7/15/2011 2:24:56.189 PM	Pass: 0
Message	7/15/2011 2:24:56.189 PM	Fail: 0
Message	7/15/2011 2:24:56.189 PM	Aborted: 0

Figure 13.9 Viewing a Test Log Report. The Complete view is shown which lists the passing and failing tests and every activity that was run as part of this test.

Running Tests Outside the DTM "Shell"

The DTM Controller provides a convenient and powerful user interface and management console to set up and run tests. But occasionally you may experience a stubborn problem with one or more tests and running these tests within the DTM shell may be too cumbersome. Or a developer may wish to exercise a particular test in his development environment away from the full DTM system. Many of the tests can be run in a manual mode outside the shell from the command line.

How to Locate Tests that Run from Outside the Shell

The DTM shell calls specific executables to run specific tests. These executables may also be run individually outside of the DTM shell. They can be found in a shared folder on the DTM controller here:

```
DTMController\Tests\x86fre\NTTest\multimediatest\avcore\audio\wdk\
```

Here is a list of the audio-related tests:

1. ac3test.exe
2. drmtest.exe
3. extfxtst.exe
4. fduplex.exe
5. fidelitytest.exe
6. fidelitytestapp.exe
7. uaatest.exe
8. wavetest.exe

For example, an APO developer should copy the APO test named extfxtst.exe to his or her local system, and use it during the development process to ensure that all APO requirements are being met.

Failures

The following section describes some common failures and what to do about them.

Jacks Not Populated

The WLK test process, for many tests, requires that all analog jacks on the system be populated with plugs. Plug-presence detection switches alert the audio driver when a jack is populated or absent. Tests such as DRM will fail if any jacks are not populated. Since the jack-insertion detection is only a physical mechanical switch that senses the presence of a plug, the plug need not be wired to a device, it need only actuate the switch, which in turn should provide an indication on the associated property panel that the jack has a plug inserted. When you are switching cables for the Audio Fidelity tests, be sure to replace any jacks after removing the test cables.

Noise Failures

Failures caused by excessive noise are the most common Audio Fidelity failures. This will show up on the Dynamic Range test and the System Activity test. From the perspective of the sensitive analog circuits in a PC, the PC environment is very hostile and noisy. Switching power supplies, motor control circuits in hard drives, video circuitry, and many other digital circuits can leak into audio circuits and be heard as buzzing noise, clicks, pops, and other offensive and unwanted noise. The physical proximity of these offending digital and switching noise sources to the analog audio circuitry, how grounding is handled, and how wiring to case-mounted jacks is done can all affect how much noise will be present. Conversely, using shielded audio cable for signal connections between the motherboard and case mounted jacks can reduce the susceptibility of these cables to picking up noise. Motherboard design principles that avoid sensitive analog signal traces being in close proximity to traces carrying digital signals will also reduce the noise contribution from these sources.

Some noise failures may be the result of incorrect mixer settings during the test run. For example, while running the render dynamic range test, only the Windows Media Player fader should be open. All other faders should be at their minimum position. Check especially the Microphone fader if there is one. If this were open during this noise test, it would allow extraneous noise that may be present in the microphone channel to be injected into the render line out and add additional noise that would likely cause a failure.

Summary

The DTM System runs a series of Audio Fidelity tests as defined by Microsoft WHQL. DTM Studio is the user interface to choose the tests to run, show progress, and indicate test results.

Tests, or jobs, can be evaluated by looking at Test Log Reports, which show every activity, what passed, what failed. After a full set of tests have been run, the resulting test logs are bundled together in a .CPK file and submitted to Microsoft. If all tests have passed, the WINQUAL site will certify the submitted drivers as passing the WHQL requirements allowing the owner of the driver to display the “Compatible with Windows” logo.