Microsoft® Research
Kinect™ for Windows® SDK beta

Programming Guide

Getting Started with the Kinect for Windows SDK Beta from Microsoft Research

Beta 1 Draft Version 1.0 – June 16, 2011

About this Guide The Kinect™ for Windows® Software Development Kit (SDK) Beta from Microsoft Research is a starter kit for applications developers. The kit makes it easier for the academic research and enthusiast communities to create rich experiences by using Kinect for Xbox 360® sensor technology on a PC that is running the Windows 7 operating system.

This SDK Beta is provided by Microsoft Research to enable researchers and programmer enthusiasts to explore the development of natural user interfaces. This SDK Beta includes an application programming interface (API) and sample code.

Important The Kinect for Windows SDK Beta from Microsoft Research is licensed only for noncommercial use. By installing, copying, or otherwise using the SDK Beta, you agree to be bound by the terms of its license. Read the license.

See Also FAQ for Kinect for Windows SDK Beta

In this guide

PART 1—Introduction to this SDK Beta
PART 2—The NUI API: An Overview
PART 3—The Audio API: An Overview
PART 4—Resources

Disclaimer: This document is provided "as-is". Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2011 Microsoft Corporation. All rights reserved.

Microsoft, DirectShow, Direct3D, DirectX, Kinect, MSDN, Visual Studio, Win32, Windows, Windows Media, Windows Vista, and Xbox 360 are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

Contents

PART 1—Introduction to This SDK Beta 4
Welcome 4
System Requirements 4
Skills for Developers 5
Downloading and Installing the SDK Beta 6
Loading the Microsoft Kinect Drivers 6
Configuring the Development Environment 7
Implementing a Managed Application 8
Implementing a C++ Application 8
Experimenting with NUI and Kinect Sensors 9
Protecting Your Kinect Sensor While You Experiment 9
Experimenting with this Beta Development Kit 11
Understanding How Kinect Sensors Work 12
Beta Testing Recommended Practices 12
Samples in the SDK Beta 12
PART 2—The NUI API: An Overview 14
Kinect for Windows Architecture 14
The NUI API 15
NUI API Initialization 15
Sensor Enumeration and Access 16
Initialization Options 17
NUI Image Data Streams: An Overview 17
Color Image Data: Quality, Formats and Bandwidth 17
Depth Data 18
Player Segmentation Data 18
Retrieving Image Information 19
NUI Skeleton Tracking 20
Retrieving Skeleton Information 20
Active and Passive Skeletal Tracking 21
NUI Transformations 22
Depth Image Space 22
Skeleton Space 22
Sensor Array and Tilt Compensation 23
Floor Determination 23
Skeletal Mirroring 24
PART 3—The Audio API: An Overview 25
About the Kinect Microphone Array 25
Kinect SDK C++ Audio API 26
Raw Audio Capture 26
KinectAudio DMO 26
Kinect SDK Managed Audio API 27
How to Use Microsoft.Speech with the Kinect Microphone Array 27
PART 4—Resources 28
Tips and Troubleshooting 28
Terminology for Kinect and NUI 29
References 30
APPENDIX 31
How to Build an Application with Visual Studio Express 31

PART 1—Introduction to This SDK Beta

Welcome

The Kinect™ for Windows® Software Development Kit (SDK) Beta from Microsoft Research is a starter kit for applications developers. The kit makes it easier for the academic research and enthusiast communities to create rich experiences by using Kinect sensor technology on computers that are running Windows 7.

Kinect for Windows SDK Beta includes:

- Drivers, for using Kinect sensor devices on a computer that is running Windows 7.
- APIs and device interfaces, together with technical documentation for developers.
- Source code samples.

System Requirements

You must run applications that are built by using the Kinect for Windows SDK Beta in a native Windows environment. You cannot run applications in a virtual machine, because the Microsoft Kinect drivers and this SDK Beta must be installed on the computer that is running the application.

Supported Operating Systems and Architectures

- Windows 7 (x86 or x64)

Hardware Requirements

- Computer with a dual-core, 2.66-GHz or faster processor
- Windows 7–compatible graphics card that supports Microsoft® DirectX® 9.0c capabilities
- 2 GB of RAM
- Kinect for Xbox 360® sensor—retail edition, which includes special USB/power cabling

Software Requirements

- Microsoft Visual Studio® 2010 Express or other Visual Studio 2010 edition
- Microsoft .NET Framework 4.0 (installed with Visual Studio 2010)
- For C++ SkeletalViewer samples:
  - DirectX Software Development Kit, June 2010 or later version
  - DirectX End-User Runtime Web Installer
- For Speech sample (x86 only):
  - Microsoft Speech Platform - Server Runtime, version 10.2 (x86 edition)
  - Microsoft Speech Platform - Software Development Kit, version 10.2 (x86 edition)
  - Kinect for Windows Runtime Language Pack, version 0.9
    (acoustic model from Microsoft Speech Platform for the Kinect for Windows SDK Beta)

Skills for Developers

You can use this SDK Beta for your personal programming interests. The SDK Beta is specifically for computer scientists and scientific researchers in academia. It is also provided for technical enthusiasts who want to creatively explore NUI possibilities with the Kinect sensor and related technologies.

Experimenting with Kinect and NUI

This SDK Beta enables exploration and experimentation with the following features:

- Skeletal tracking for the image of one or two persons who are moving within the Kinect sensor's field of view.
- XYZ-depth camera for accessing a standard color camera stream plus depth data to indicate the distance of the object from the Kinect sensor.
- Audio processing for a four-element microphone array with acoustic noise and echo cancellation, plus beam formation to identify the current sound source and integration with the Microsoft.Speech speech recognition API.

Related concepts and programming approaches are explored in this programming guide.

Application Development

To take advantage of the experimental capabilities and features in this SDK Beta, developers should be familiar with the following:

- Development environment and languages
  - Visual Studio 2010

    To work with the SDK samples and to develop your own applications with the SDK, you can use Visual Studio C# 2010 Express, Visual Studio C++ 2010 Express, or any other version of Visual Studio 2010. Visual Studio Express is available at no cost from the Microsoft website. If you are new to Visual Studio, see the Visual Studio 2010 Getting Started Guide.
  - C# or C++ languages

    Samples are available in both C# and C++. For information about programming in these languages, see Visual C# Developer Center and Visual C++ Developer Center.
- Application development for Windows 7

  This SDK Beta takes advantage of the functions and features of the SDK for Windows 7. If you are new to Windows development, see Beginner Developer Learning Center for Windows Development.

Note This SDK should not be used to develop prototype applications with the intent of porting those applications to the Xbox 360 console. There are numerous architectural, performance, and behavioral differences between Xbox 360 and Windows. These differences affect how the Kinect technology is implemented for Windows 7. As a result, design decisions and performance characteristics are quite often very different between the two platforms. Microsoft recommends using Xbox 360 development kits (XDK hardware and software) to create Xbox 360 applications

Downloading and Installing the SDK Beta

This section describes how to install the SDK Beta and begin experimenting with the Kinect sensor on Windows.

You must download and install the SDK Beta on any system where you want to build or run applications that use the Kinect sensor on Windows.

To prepare to install the Kinect for Windows SDK Beta

1. Obtain the latest updates for Windows 7 from Microsoft Update.
2. Make sure that the Kinect device is not plugged in to the USB port on your computer.
3. Remove any other drivers you might have previously installed for the Kinect device, including any earlier versions of the Microsoft Kinect drivers or drivers from any other source. To do this:
    - In **Programs and Features** in Control Panel, click the name of any previous driver, and then click **Uninstall**.
    - If the driver is not listed in **Programs and Features**, follow the directions that the driver distributor provides for removing the driver.

    Caution If another driver for the Kinect device is installed on your PC, the Microsoft Kinect drivers might not install correctly and will not work correctly.
4. Make sure all required software is installed on your computer, as described earlier in "System Requirements."
5. Close Visual Studio before you install the SDK Beta.

    Note To ensure proper setting of the Path MSRKINECTSDK environment variables that the SDK Beta requires, you must exit Visual Studio before installation and then restart it after installation.

To obtain and install the Kinect for Windows SDK Beta

1. On the Kinect for Windows SDK Beta download page, click the **Download** option for your computer (x86 or x64).
2. To start the SDK installation immediately, click **Run**.
   To save the download to your computer for installation at a later time, click **Save**.
3. Follow the Setup wizard prompts to complete installation.

Important You must install the Kinect for Windows SDK Beta on each PC system on which you want to run applications that you build by using this SDK Beta.

Loading the Microsoft Kinect Drivers

The Kinect sensor is a physical device that contains cameras and a microphone array. The sensor connects to a PC by a USB cable although it is not, strictly speaking, a USB device. The Kinect sensor uses a USB connector, but has somewhat different firmware than standard USB devices.

When you download the SDK Beta, the installation package includes the Microsoft Kinect drivers. When the Microsoft Kinect drivers are installed on a Windows-based PC, a Kinect sensor that is plugged into the PC's USB port appears to applications as a multicomponent USB device.

To load the Microsoft Kinect drivers

1. Plug the power supply for your Kinect sensor into an external power source.
2. Plug in the Kinect sensor to a USB port on your PC and wait for Windows to recognize the sensor's components.
3. All the drivers, including audio, will load seamlessly.

To verify the driver installation

1. You should see a blinking green LED on the Kinect sensor.
2. In **Device Manager** in Control Panel, look for the following nodes under **Microsoft Kinect**:
   - Microsoft Kinect Audio Array Control
   - Microsoft Kinect Camera
   - Microsoft Kinect Device
3. Under **Sound, Video and Game Controllers** in **Device Manager**, the Kinect sensor's microphone array should appear
   - Kinect USB Audio

Important You must install the Kinect for Windows SDK Beta—which also installs the Microsoft Kinect drivers—before you plug the Kinect device into a USB port on your computer. If you plug in the device before installing the drivers, **Microsoft Kinect** will not appear in Device Manager. If you have more than one Kinect sensor, make sure that you connect them to different USB hubs.

Configuring the Development Environment

Visual Studio Express Edition for either C++ or C# includes an integrated development environment (IDE) in which you can code, debug, and test your applications.

This section provides notes for configuring your development environment to work with the Kinect for Windows SDK Beta.

New to Visual Studio Express? If you are new to Visual Studio Express as a development environment, the following information on the MSDN® website will help you become familiar with these development tools:

- For a brief look at how to configure Visual Studio Express and then build and test a new project, see the appendix to this guide, "How to Build an Application with Visual Studio Express."
- Visit the Visual C# Developer Center and Visual C++ Developer Center, where you can find many tutorials, samples, and videos to help you get started.

General Notes

- Use Visual Studio 2010 Express or another Visual Studio 2010 edition.
- Specify an **x86** (Win32) platform target.

  Important Do not specify x64 or Any CPU as the platform target. This SDK Beta supports only x86 applications. The 32-bit and 64-bit installation packages for this SDK Beta include Kinect for Windows drivers for x86 and x64 systems, respectively, but both packages include only x86 libraries. All applications must therefore target only the x86 platform.

- Applications that are built with this SDK Beta must run in a native Windows environment. You cannot run applications in a virtual machine, because the Microsoft Kinect drivers and this SDK Beta must be installed on the computer that is running the application.

Implementing a Managed Application

Follow these basic steps for implementing a managed application.

To implement a C# application

1. Reference Microsoft.Research.Kinect.dll.

   This assembly is in the global assembly cache (GAC) and appears on the .NET tab of the Add Reference dialog box. This DLL calls unmanaged functions from managed code.

2. Include **using** directives for the following namespaces:

   For the NUI API, include:

   ```
   using Microsoft.Research.Kinect.Nui
   ```

   For the Audio API, include:

   ```
   using Microsoft.Research.Kinect.Audio
   ```

Implementing a C++ Application

Follow these basic steps for implementing an unmanaged application.

To implement a C++ application

1. Do not compile C++ projects with w_char as a separate type.
2. To use the NUI API, include MSR_NuiApi.h.

   Location: Program Files\Microsoft Research KinectSDK\inc

3. To use the Kinect Audio API, include MSRKinectAudio.h.

   Location: Program Files\Microsoft Research KinectSDK\inc

4. Link to MSRKinectNUI.lib.

   Location: Program Files\Microsoft Research KinectSDK\lib

5. Ensure that the Kinect SDK DLLs are on your path when you run your project.

   Location: \Program Files\Microsoft Research KinectSDK

The following table describes the files to include in your projects.

In ..\Program Files (x86)\Microsoft Research KinectSDK\inc	
MSR_NuiApi.h	Aggregates all NUI API headers and defines basic initialization and access functions—NuiInitialize, NuiShutdown, MSR_NuiXxx, and INuiInstance: • Enumerate devices • Access multiple devices

In ..\Program Files (x86)\Microsoft Research KinectSDK\inc\	
MSR_NuiImageCamera.h	Defines the API for the NUI image and camera services—NuiCameraXxx and NuiImageXxx: • Adjust camera angle and elevation • Open streams and read image frames
MSR_NuiProps.h	Defines the APIs for the NUI properties enumerator.
MSR_NuiSkeleton.h	Defines API for NUI Skeleton—NuiSkeletonXxx and NuiTransformXxx: • Enable/disable skeleton tracking • Get skeleton data • Transform skeleton data for smoother rendering
MSRKinectAudio.h	Defines the Audio API, including the ISoundSourceLocalizer interface that returns the beam direction and source location.
NuiImageBuffer.h	Defines a frame buffer for effects that are similar to DirectX 9 textures.

Experimenting with NUI and Kinect Sensors

This SDK Beta is intended for an audience of experienced developers who want to experiment with Kinect sensors as input and imaging devices on Windows-based PCs. Many early-adopters are eager to use Kinect sensors to explore possibilities for natural user interfaces (NUI). The SDK is not intended for use in consumer environments and is not licensed for production use. Use of this SDK is expected to be in a laboratory or test-bench setting or in a classroom.

This guide provides the following brief notes to get you started:

- Experimenting with this Beta development kit.
- Protecting your Kinect sensor while you experiment.
- Scenarios and ranges for the Kinect sensor array.

Feedback Request The Microsoft teams that collaborated to deliver this early SDK Beta want to gain feedback from developers about how to develop software that takes advantage of Kinect sensors in the Windows environment. Please use the forums to provide feedback about your experimentation and research when you use this SDK Beta.

Protecting Your Kinect Sensor While You Experiment

The following tips can help you protect your Kinect device while you experiment with possibilities by using this SDK Beta.

Standalone Sensor

- Plug the Kinect sensor's power supply into an external power source.

  With USB power only, the sensor is minimally functional and the LED lights up. However, to be fully functional, the sensor must be connected to an external power source.

- No tools are required for calibration of audio and video.

Controlling the Fan—Avoiding Overheating

- The Kinect sensor is protected from overheating by a fan. It is controlled by the sensor's firmware, which turns off the camera at 90 degrees Celsius.
- There is no software interface for applications to control the fan.

Controlling the Sensors

- All control of the Kinect sensor is through the APIs that are provided in this SDK Beta.

  Caution If you use any other mechanisms to control the Kinect sensor, the sensor could become irreparably damaged.

- The tilt mechanism in the sensor array is not rated for frequent use. Your code should not make calls to change the camera angle more than once per second or more than 15 times in any 20-second period.

  Warning You should tilt the Kinect sensor as few times as possible, to minimize wear on the camera and to minimize tilting time. The camera motor is not designed for constant or repetitive movement, and attempts to use it that way may cause degradation of motor function. This Beta SDK limits the rate at which applications can tilt the sensor, to protect the Kinect hardware. If the application tries to tilt the sensor too frequently, the runtime imposes a short lockout period during which any further calls return an error code.

Sensor Placement

When using the Kinect sensor with a Windows PC, make sure to place the sensor on a stable surface, in a location where it will not fall or be struck during use. In addition:

- Do not place the sensor on or in front of a speaker or a surface that vibrates or makes noise.
- Keep the sensor out of direct sunlight.
- Do not use the sensor near any heat sources.
- Use the sensor within its specified operating temperature range of 41 to 95 degrees Fahrenheit (5 to 35 degrees Celsius).

  If the sensor is exposed to an environment outside its prescribed range, turn it off and allow the temperature to stabilize within the specified range before you use the sensor again.

  Caution Adjust the sensor location only by moving the base. Do not adjust the sensor viewing angle by hand, by tilting the sensor on its base. After setup is complete, let the sensor motors adjust the viewing angle, to avoid the risk of damaging your sensor.

Ranges for the Kinect Sensor Array

The current generation of the Kinect sensor is designed for game-playing and similar scenarios. While you are experimenting with the Kinect sensor and NUI scenarios, remember the following:

- **Standing Scenarios Only** This SDK Beta enables skeletal viewing for standing scenarios only, not seated figures.
- **Range = 4 to 11 Feet** For the depth and skeletal views, the figures should stand between 4 and 11 feet from the sensor.

Important If you stand too close to the sensor or experiment while the sensor is on your desktop, the result is typically empty or noisy depth views or skeletal views.

The Kinect sensor ranges and the sensor array specifications are shown in the following tables.

Playable Ranges for the Kinect for Windows Sensor

Sensor item	Playable range
Color and depth stream	4 to 11.5 feet (1.2 to 3.5 meters)
Skeletal tracking	4 to 11.5 feet (1.2 to 3.5 meters)

Kinect Sensor Array Specifications

Sensor item	Specification range
Viewing angle	43° vertical by 57° horizontal field of view
Mechanized tilt range (vertical)	±28°
Frame rate (depth and color stream)	30 frames per second (FPS)
Resolution, depth stream	QVGA (320 × 240)
Resolution, color stream	VGA (640 × 480)
Audio format	16-kHz, 16-bit mono pulse code modulation (PCM)
Audio input characteristics	A four-microphone array with 24-bit analog-to-digital converter (ADC) and Kinect-resident signal processing such as echo cancellation and noise suppression

Experimenting with this Beta Development Kit

To enable early exploration, this SDK Beta provides the following basics for developers:

- An easy-to-use programming interface that is supported by the Microsoft Kinect drivers and the APIs in this SDK.
- APIs to control audio and video data streams and skeletal tracking in the Windows environment.
- Delivery of multiple media streams with minimal software latency across various video, CPU, and device variables.

The Kinect sensor was designed specifically for use with Xbox 360 in living-room entertainment applications. This SDK Beta opens up the potential to repurpose the Kinect sensor for use in PC applications in other contexts, but this repurposing does not change the physical design and capabilities of the sensor.

This SDK Beta is based on the Xbox 360 software, but this SDK Beta does not describe simply how to port software from one computing environment to another. Numerous architectural, performance, and behavioral differences exist between Xbox 360 and Windows. These differences affect how Windows support has been implemented for Kinect technologies. As a result, design decisions and performance characteristics are quite often very different between the two platforms. For example, the Kinect audio solution on Windows is completely different from the solution for the Xbox 360 console. Support for the four-microphone array Kinect audio support also requires an extension to the built-in Windows 7 audio capabilities.

This section provides some technical information and recommended practices for working with preview software such as this Beta SDK.

Understanding How Kinect Sensors Work

The Microsoft Kinect sensor was developed and optimized to work with Xbox 360 consoles. The Kinect hardware and firmware have not been redesigned to work on the Windows platform. Because the Kinect sensor is a brand new, unsupported device in the Windows environment, consider the following:

- The multifunction devices that are exposed in the Kinect sensor are not standard USB devices. Therefore:
  - Comprehensive Windows Plug and Play capabilities are not available.
  - Windows power management is not implemented.

  As a result, your system might behave in unpredictable ways in sleep transitions or with surprise device removal.
- Depth camera, multiple-microphone arrays, and other capabilities are unique to the Kinect sensor. Some effects might seem counter-intuitive when you attempt to work with the data streams.

  As you experiment with controlling Kinect inputs and outputs, consider reading more about the unique capabilities of the Kinect sensor. For technical background information, see the research papers in “Resources,” later in this guide.

Beta Testing Recommended Practices

When you work with any beta software such as this SDK Beta, consider these tips:

- Control the factors that can influence your development work, so that you can focus on developing and debugging your own application:
  - Do not connect your Kinect device to a USB hub that is shared by other devices.
  - Disconnect other USB media devices while you experiment with your Kinect application.
  - Avoid running other beta software or drivers, which can interact with your SDK Beta testing in unpredictable ways.
- This SDK Beta has not been tested in the following ways:
  - Developing with other SDKs and frameworks, such as XNA Game Studio.
  - Interoperating with productivity applications or other software that might attempt to use the multimedia capabilities of the Kinect device, such as voice-over-IP (VOIP) applications that depend on a microphone.

Samples in the SDK Beta

The Windows for Kinect SDK Beta includes several sample programs to help you experiment with the Kinect sensor on Windows. The samples demonstrate basic NUI features and coding techniques, and are provided to help you explore fundamental concepts for using this SDK Beta. These samples are installed in your \Users\Public\Documents\Microsoft Research KinectSDK Samples\ folder.

Detailed “walkthrough” documentation for several of these samples can be found on the SDK Beta website, including:

SkeletalViewer Walkthrough—Capturing Data with the NUI API (C++ and C#)
Uses the NUI API to access and render data from the Kinect sensor’s depth and video cameras as depth, video, and skeletal images on the screen. The managed SkeletalViewer sample uses

Windows Presentation Foundation (WPF) to render captured images, and the native application uses DirectX and the Windows graphic display interface (GDI).

AudioCaptureRaw Walkthrough—Capturing the Raw Audio Stream (C++)
Uses the Windows Audio Session API (WASAPI) to capture the raw audio stream from the Kinect sensor's four-element microphone array and write it to a .wav file.

MFAudioFilter Walkthrough—Capturing Streams with a Media Foundation Audio Filter (C++)
Shows how to capture an audio stream from the Kinect sensor's microphone array by using the KinectAudio DirectX Media Object (DMO) in filter mode in a Windows Media® Foundation topology.

MicArrayEchoCancellation Walkthrough—Capturing Audio Streams with Acoustic Echo Cancellation and Beamforming (C++)
Shows how to use the KinectAudio DMO as a DirectShow® source to access the Kinect sensor's microphone array. This sample uses acoustic echo cancellation to record a high-quality audio stream and beamforming and source localization to determine the selected beam and the direction of the sound source.

RecordAudio Walkthrough—Recording an Audio Stream and Monitoring Direction (C#)
Demonstrates how to capture an audio stream from the Kinect sensor's microphone array and how to monitor the selected beam and the direction of the sound source.

Speech Walkthrough—Recognizing Voice Commands (C#)
Demonstrates how to use the Kinect sensor's microphone array with Microsoft.Speech API to recognize voice commands.

PART 2—The NUI API: An Overview

Kinect for Windows Architecture

The Kinect for Windows SDK Beta provides a sophisticated software library and tools to help developers use the rich form of Kinect-based natural input, sensing and reacting to real-world events.

The Kinect sensor and associated software library interact with your application, as shown in Figure 1.

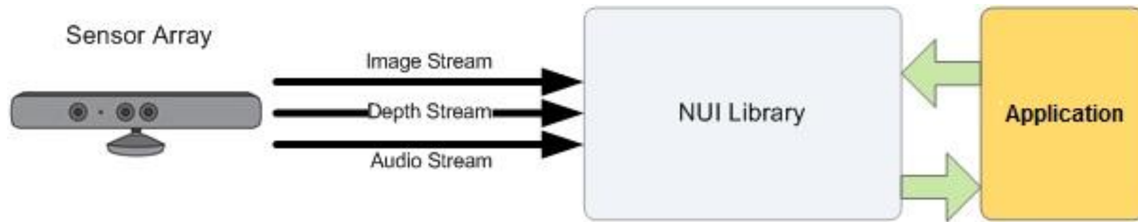


Figure 1. Hardware and software interaction with an application

The components of the Kinect for Windows SDK Beta are shown in Figure 2.

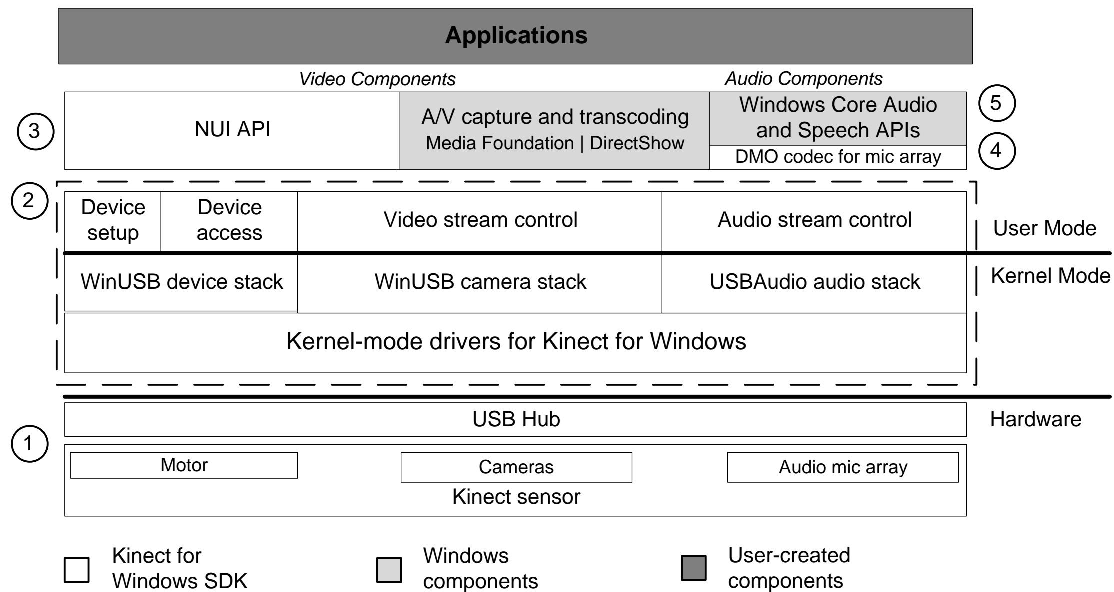


Figure 2. Kinect for Windows SDK Beta architecture

Components for Kinect for Windows shown in Figure 2 include the following:

1. **Kinect hardware**
   The hardware components, including the Kinect sensor and the USB hub, through which the sensor is connected to the computer.

2. **Microsoft Kinect drivers**
   The Windows 7 drivers for the Kinect sensor, which are installed as part of the SDK Beta setup process as described in this document. The Microsoft Kinect drivers support:
   - The Kinect sensor's microphone array as a kernel-mode audio device that you can access through the standard audio APIs in Windows.
   - Streaming image and depth data.
   - Device enumeration functions that enable an application to use more than one Kinect sensor that is connected to the computer.
3. **NUI API**
   A set of APIs that retrieves data from the image sensors and controls the Kinect devices.
4. **KinectAudio DMO**
   The Kinect DMO that extends the microphone array support in Windows 7 to expose beamforming and source localization functionality.
5. **Windows 7 standard APIs**
   The audio, speech, and media APIs in Windows 7, as described in the Windows 7 SDK and the Microsoft Speech SDK.

The NUI API

The NUI API is the core of the Kinect for Windows API. It supports fundamental image and device management features, including the following:

- Access to the Kinect sensors that are connected to the computer.
- Access to image and depth data streams from the Kinect image sensors.
- Delivery of a processed version of image and depth data to support skeletal tracking.

Note This SDK Beta includes C++ and C# versions of the SkeletalViewer sample. SkeletalViewer shows how to use the NUI API in an application to capture data streams from the NUI Image camera, use skeletal images, and process sensor data. For more information, see "Skeletal Viewer Walkthrough" on the SDK Beta website.

NUI API Initialization

The Microsoft Kinect drivers support the use of multiple Kinect sensors on a single computer. The NUI API includes functions that enumerate the sensors, so that you can determine how many sensors are connected to the machine, get the name of a particular sensor, and individually open and set streaming characteristics for each sensor.

Although the SDK Beta supports the use of multiple Kinect sensors by an application, only one application can use each sensor at any given time.

Sensor Enumeration and Access

C++ and managed code applications enumerate the available Kinect sensors, open a sensor, and initialize the NUI API in one of the following ways:

To initialize the NUI API and use a Kinect sensor in a C++ application

If the application uses only one Kinect sensor:

1. Call **NuiInitialize**. This function initializes the first instance of the Kinect sensor device on the system.
2. Call other **NuiXxx** functions to stream image and skeleton data and manage the cameras.
3. Call **NuiShutdown** when use of the Kinect sensor is complete.

If the application uses more than one sensor:

1. Call **MSR_NuiDeviceCount** to determine how many sensors are available.
2. Call **MSR_NuiCreateInstanceByIndex** to create an instance for each sensor that the application uses. This function returns an **INuiInstance** interface pointer for the instance.
3. Call **INuiInstance::NuiInitialize** to initialize the NUI API for the sensor.
4. Call other methods on the **INuiInstance** interface to stream image and skeleton data and manage the cameras.
5. Call **INuiInstance::NuiShutdown** on a sensor instance to close the NUI API when use of that sensor is complete.
6. Call **MSR_NuiDestroyInstance** to destroy the instance.

To initialize the NUI API and use a Kinect sensor in managed code

If the application uses only one Kinect sensor:

1. Create a new **Runtime** object and leave the parameter list empty, as in the following C# code:

   ```
   nui = new Runtime();
   ```

   This constructor creates an object that represents the first instance of the Kinect sensor device on the system.
2. Call **Runtime.Initialize** to initialize the NUI API for the sensor.
3 Call additional methods in the managed interface to stream image and skeleton data and manage the cameras.
4. Call **Runtime.Shutdown** when use of the Kinect sensor is complete.

If the application uses more than one sensor:

1. Call **MSR_NuiDeviceCount** to determine how many sensors are available.
2. Create a new **Runtime** object and pass the index of a sensor, as in the following C# code:

   ```
   nui = new Runtime(index);
   ```

   This constructor creates an object that represents a particular instance of the Kinect sensor device on the system.
3. Call **Runtime.Initialize** to initialize the NUI API for that device instance.

4 Call additional methods in the managed interface to stream image and skeleton data and manage the cameras.

5. Call **Runtime.Shutdown** when use of that device instance is complete.

Initialization Options

The NUI API processes data from the Kinect sensor through a multi-stage pipeline. At initialization, the application specifies the subsystems that it uses, so that the runtime can start the required portions of the pipeline. An application can choose one or more of the following options:

- **Color.** The application streams color image data from the sensor.
- **Depth.** The application streams depth image data from the sensor.
- **Depth and player index.** The application streams depth data from the sensor and requires the player index that the skeleton tracking engine generates.
- **Skeleton.** The application uses skeleton position data.

These options determine the valid stream types and resolutions for the application. For example, if an application does not indicate at NUI API initialization that it uses depth, it cannot later open a depth stream.

NUI Image Data Streams: An Overview

The NUI API provides the means to modify settings for the Kinect sensor array, and it enables you to access image data from the sensor array.

Stream data is delivered as a succession of still-image frames. At NUI initialization, the application identifies the steams it plans to use. It then opens those streams with additional stream-specific details, including stream resolution, image type, and the number of buffers that the runtime should use to store incoming frames. If the runtime fills all the buffers before the application retrieves and releases a frame, the runtime discards the oldest frame and reuses that buffer. As a result, it is possible for frames to be dropped. An application can request up to four buffers; two is adequate for most usage scenarios.

An application has access to the following kinds of image data from the sensor array:

- Color data
- Depth data
- Player segmentation data

Color Image Data: Quality, Formats and Bandwidth

Color image data is available at two quality levels and in two different formats:

- Quality level determines how quickly data is transferred from the Kinect sensor array to the PC.
- The available color formats determine whether the image data that is returned to application code is encoded as RGB or YUV.

Image Quality and Bandwidth The sensor array uses a USB connection to pass data to the PC, and that connection provides a given amount of bandwidth. Your choice of image data quality lets you tune how that bandwidth is used. High-quality images send more data per frame and update less frequently, whereas regular quality images update more frequently with some loss in image quality due to compression:

- At regular quality, the Bayer color image data that the sensor returns at 1280x1024 is compressed and converted to RGB before transmission to the runtime. The runtime then decompresses the data before it passes the data to your application. The use of compression makes it possible to return color data at frame rates as high as 30 FPS, but the algorithm that is used leads to some loss of image fidelity.
- At high quality, color image data is not compressed in the sensor—it is transmitted to the runtime as originally captured by the sensor. Because the data is not compressed, more data must be transmitted per frame and the maximum frame rate is no more than 15 FPS. Also, the uncompressed data requires that the NUI system allocate larger buffers.

You must evaluate what kind of color data is most appropriate in your application: high-quality data in larger buffers at lower frame rates or data at higher frame rates with some loss of fidelity and a smaller memory footprint.

Formats Color data is available in the following two formats:

- **RGB color** provides 32-bit, linear X8R8G8B8-formatted color bitmaps, in sRGB color space. To work with RGB data, an application should specify a color or color_YUV image type when it opens the stream.
- **YUV color** provides 16-bit, gamma-corrected linear UYVY-formatted color bitmaps, where the gamma correction in YUV space is equivalent to sRGB gamma in RGB space. Because the YUV stream uses 16 bits per pixel, this format uses less memory to hold bitmap data and allocates less buffer memory when you open the stream. To work with YUV data, your application should specify the raw YUV image type when it opens the stream. YUV data is available only at the 640x480 resolution and only at 15 FPS.

Both color formats are computed from the same camera data, so that the YUV data and RGB data represent the same image. Choose the data format that is most convenient given your application's implementation.

Depth Data

The depth data stream provides frames in which the high 13 bits of each pixel give the distance, in millimeters, to the nearest object at that particular x and y coordinate in the depth sensor's field of view. The following depth data streams are available:

- Frame size of 640×480 pixels
- Frame size of 320×240 pixels
- Frame size of 80×60 pixels

Applications can process data from a depth stream to support various custom features, such as tracking users' motions or identifying background objects to ignore during application play.

Player Segmentation Data

In the Kinect for Windows SDK Beta, the Kinect system processes sensor data to identify two human figures in front of the sensor array and then creates the Player Segmentation map. This map is a bitmap in which the pixel values correspond to the player index of the person in the field of view who is closest to the camera, at that pixel position.

Although the player segmentation data is a separate logical stream, in practice the depth data and player segmentation data are merged into a single frame:

- The 13 high-order bits of each pixel represent the distance from the depth sensor to the closest object, in millimeters.
- The 3 low-order bits of each pixel represent the player index of the tracked player who is visible at the pixel's x and y coordinates. These bits are treated as an integer value and are not used as flags in a bit field.

A player index value of zero indicates that no player was found at that location. Values one and two identify players.

Applications commonly use player segmentation data as a mask to isolate specific users or regions of interest from the raw color and depth images.

Retrieving Image Information

Application code gets the latest frame of image data by calling a frame retrieval method and passing a buffer. If the latest frame of data is ready, it is copied into the buffer. If your code requests frames of data faster than new frames are available, you can choose whether to wait for the next frame or to return immediately and try again later. The NUI Image Camera API never provides the same frame of data more than once.

Applications can use either of the following two usage models:

- **Polling Model** The *polling* model is the simplest option for reading data frames.

  First, the application code requests a frame and specifies how long to wait for the next frame of data (between 0 and an infinite number of milliseconds). The request method returns when a new frame of data is ready or when the wait time expires, whichever comes first. Specifying an infinite wait causes the call for frame data to block and to wait as long as necessary for the next frame.

  When the request returns successfully, the new frame is ready for processing. If the time-out value is set to zero, the application code can poll for completion of a new frame while it performs other work on the same thread.

  A native C++ application calls **NuiImageGetNextFrame** to poll for color and depth frames. Managed code calls **ImageStream.GetNextFrame**.

- **Event Model** The *event* model supports the ability to integrate retrieval of a skeleton frame into an application engine with more flexibility and more accuracy.

  In this model, C++ application code passes an event handle to **NuiImageStreamOpen**. When a new frame of image data is ready, the event is signaled. Any waiting thread wakes and gets the frame of skeleton data by calling **NuiImageGetNextFrame**. During this time, the event is reset by the NUI Image Camera API.

  Managed code uses the event model by hooking a **Runtime.DepthFrameReady** or **Runtime.ImageFrameReady** event to an appropriate event handler. When a new frame of data is ready, the event is signaled and the handler runs and calls **ImageStream.GetNextFrame** to get the frame.

NUI Skeleton Tracking

The NUI Skeleton API provides information about the location of up to two players standing in front of the Kinect sensor array, with detailed position and orientation information.

The data is provided to application code as a set of points, called *skeleton positions*, that compose a skeleton, as shown in Figure 3. This skeleton represents a user's current position and pose. Applications that use skeleton data must indicate this at NUI initialization and must enable skeleton tracking.

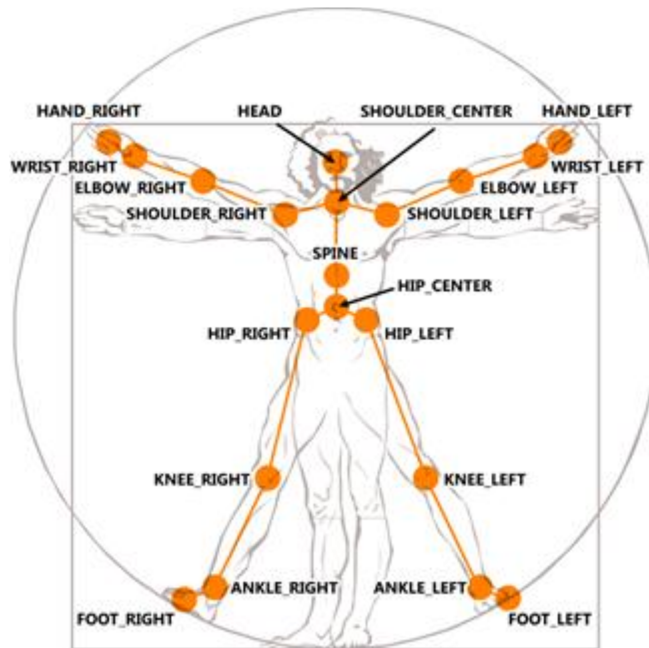


Figure 3. Skeleton joint positions relative to the human body

Retrieving Skeleton Information

Application code gets the latest frame of skeleton data in the same way that it gets a frame of image data: by calling a frame retrieval method and passing a buffer. Applications can use either a polling model or an event model, in the same way as for image frames. An application must choose one model or the other; it cannot use both models simultaneously.

To use the polling model:

- A native C++ application calls **NuiSkeletonGetNextFrame** to retrieve a skeleton frame.
- Managed code calls **SkeletonEngine.GetNextFrame**.

To use the event model:

- C++ application code passes an event handle to **NuiSkeletonTrackingEnable**.

  When a new frame of skeleton data is ready, the event is signaled. Any waiting thread wakes and gets the frame of skeleton data by calling **NuiSkeletonGetNextFrame**. During this time, the event is reset by the NUI Skeleton API.
- Managed code uses the event model by hooking a **Runtime.SkeletonFrameReady** event to an appropriate event handler.

When a new frame of skeleton data is ready, the event is signaled and the handler runs and calls **SkeletonEngine.GetNextFrame** to get the frame.

The skeletal tracking engine processes depth frame data to calculate the normal to gravity and the floor clipping plane, which are described in "Sensor Array and Tilt Compensation," later in this document. If the application indicates at initialization that it uses skeleton tracking, the skeletal tracking engine signals a skeleton frame each time it processes the depth data, whether or not a skeleton currently appears in the frame. Applications that use the normal to gravity and floor clipping plane values can retrieve the skeleton frame. The returned skeleton frame includes the timestamp of the corresponding depth image so that applications can match skeleton data with depth image data.

Active and Passive Skeletal Tracking

The skeletal tracking engine provides full skeletal tracking for one or two players in the sensor's field of view. When a player is actively tracked, calls to get the next skeleton frame return complete skeleton data for the player. Passive tracking is provided automatically for up to four additional players in the sensor's field of view. When a player is being tracked passively, the skeleton frame contains only limited information about that player's position. By default, the first two skeletons that the skeletal tracking system finds are actively tracked, as shown in Figure 4.

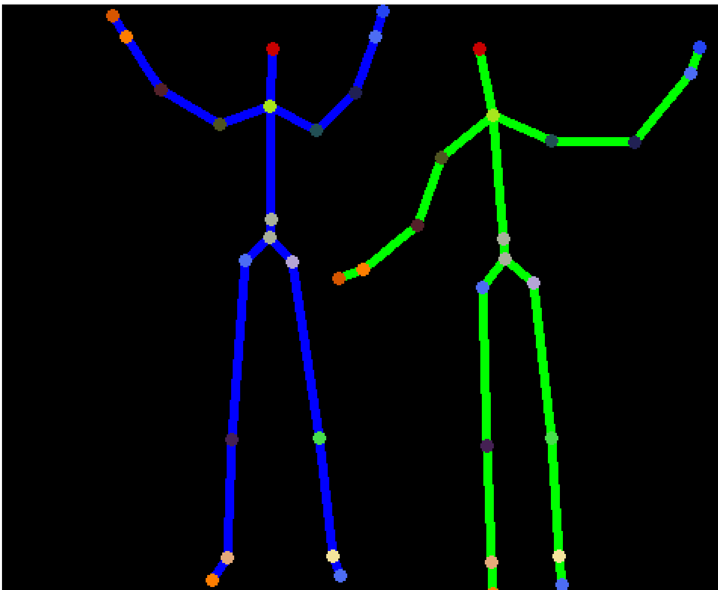


Figure 4. Active tracking for two players

The runtime returns skeleton data in a skeleton frame, which contains an array of skeleton data structures, one for each skeleton that the skeletal tracking system recognized. Not every skeleton frame contains skeleton data. When skeleton tracking is enabled, the runtime signals a skeleton event every time it processes a depth frame, as described in the previous section.

For all returned skeletons, the following data is provided:

- The current tracking state of the associated skeleton.
    - For skeletons that are passively tracked, this value indicates position-only tracking.
    - For an actively tracked skeleton, the value indicates skeleton-tracked.

- A unique tracking ID that remains assigned to a single player as that player moves around the screen.

  The tracking ID is guaranteed to remain consistently applied to the same player for as long as he or she remains in the field of view. A given tracking ID is guaranteed to remain at the same index in the skeleton data array for as long as the tracking ID is in use. If the tracking ID of the skeleton at a particular index in the array changes, one of two things has happened: Either the tracked player left the field of view and tracking started on another player in the field of view, or the tracked player left the field of view, then returned, and is now being tracked again.
- A position (of type **Vector4**) that indicates the center of mass for that player. This value is the only available positional value for passive players.
- For the actively tracked players, returned data also includes the current full skeletal data.
- For the passively tracked players, returned data includes only basic positional and identification data, and no skeletal data.

NUI Transformations

This section provides a brief overview of the various coordinate systems that are used with skeleton tracking and the API support that is provided for transformations between these spaces.

Depth Image Space

Image frames of the depth map are 640x480, 320×240, or 80x60 pixels in size, with each pixel representing the distance, in millimeters, to the nearest object at that particular x and y coordinate. A pixel value of 0 indicates that the sensor did not find any objects within its range at that location. The x and y coordinates of the image frame do not represent physical units in the room, but rather pixels on the depth imaging sensor. The interpretation of the x and y coordinates depends on specifics of the optics and imaging sensor. For discussion purposes, this projected space is referred to as the *depth image space*.

Skeleton Space

Player skeleton positions are expressed in x, y, and z coordinates. Unlike the coordinate of depth image space, these three coordinates are expressed in meters. The x, y, and z axes are the body axes of the depth sensor. This is a right-handed coordinate system that places the sensor array at the origin point with the positive z axis extending in the direction in which the sensor array points. The positive y axis extends upward, and the positive x axis extends to the left (with respect to the sensor array), as shown in Figure 5. For discussion purposes, this expression of coordinates is referred to as the *skeleton space*.

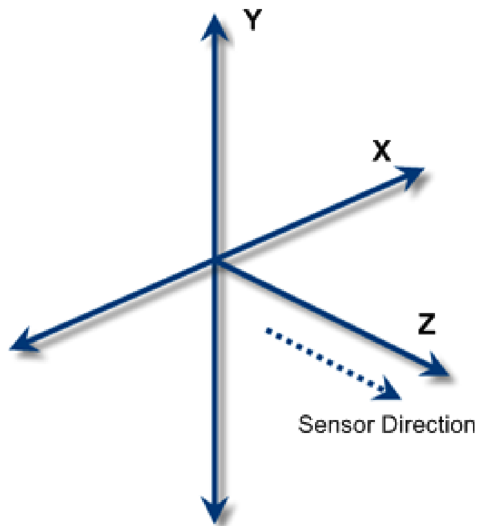


Figure 5. Skeleton-space coordinate system for the sensor array

Sensor Array and Tilt Compensation

Placement of the sensor array affects the images that that the camera records. For example, the camera might be placed on a surface that is not level or the sensor array might be vertically pivoted to optimize the sensor's field of view. In these cases, the y-axis of the skeleton space is usually not perpendicular to the floor or parallel with gravity. In the resulting images, people standing up straight could appear to be leaning.

For these reasons, each skeleton frame contains a vector that describes the normal to gravity value. The value is a measurement from an internal three-axis accelerometer that has been calibrated to align with the image sensor. In the absence of motion, the accelerometer measures a 1-g normal force to gravity. This can be treated as an upward vector.

The normal force to gravity can be found in the **vNormalToGravity** member of the NUI_SKELETON_FRAME structure in the native interface and in the **SkeletonFrame.NormalToGravity** field in the managed interface.

Floor Determination

Each skeleton frame also contains a floor clip plane vector, which contains the coefficients of an estimated floor plane equation. The skeleton tracking system updates this estimate for each frame and uses it as a clipping plane for removing the background and segmentation of players. The general plane equation is:

$$Ax + By + Cz + D = 0$$

where:

A = vFloorClipPlane.x
B = vFloorClipPlane.y
C = vFloorClipPlane.z
D = vFloorClipPlane.w

The equation is normalized so that the physical interpretation of *D* is the height of the camera from the floor, in meters. It is worth noting that the floor might not always be visible. In this case, the floor clip plane is a zero vector.

The floor clip plane can be found in the **vFloorClipPlane** member of the NUI_SKELETON_FRAME structure in the native interface and in the **SkeletonFrame.FloorClipPlane** field in the managed interface.

Skeletal Mirroring

By default, the skeleton system always mirrors the user who is being tracked. For applications that use an avatar to represent the user, such mirroring could be desirable if the avatar is shown facing into the screen. However, if the avatar faces the user, mirroring would present the avatar as backwards. Depending on its requirements, an application can create a transformation matrix to mirror the skeleton and then apply the matrix to the points in the array that contains the skeleton positions for that skeleton. The application is responsible for choosing the proper plane for reflection.

PART 3—The Audio API: An Overview

About the Kinect Microphone Array

Microsoft Kinect audio features are supported by a microphone array. In general, a microphone array consists of a set of microphones—typically four—that are usually several centimeters apart and arranged in a linear or L-shaped pattern. An array of multiple independent microphones has the following significant advantages over a single microphone:

- **Improved audio quality** Microphone arrays can support more sophisticated and effective noise suppression and automatic echo cancellation (AEC) algorithms than are possible with a single microphone.
- **Beamforming and source localization** By using the fact that the sound from a particular audio source arrives at each microphone in the array at a slightly different time, beamforming allows applications to determine the direction of the audio source and use the microphone array as a steerable directional microphone.

For a detailed discussion of microphone arrays and microphone array support in Windows, see the white papers "Microphone Array Support in Windows Vista" and "How to Build and Use Microphone Arrays for Windows Vista" on the MSDN website. For a general conceptual discussion, see the "Beamforming" topic on Wikipedia.

The Kinect sensor includes a four-element linear microphone array, which uses 24-bit ADC and provides local signal processing, including echo cancellation and noise suppression. Applications that are created with this SDK Beta can use the Kinect microphone array for the following:

- High-quality audio capture
- Beamforming and source localization

  The MSRKinectAudio DMO includes built-in algorithms that control the "beam" and provide the source direction to applications.

- Speech recognition

  With this SDK Beta, applications can use the Kinect microphone array as an input device for the Microsoft.Speech speech recognition API.

The Kinect for Windows SDK Beta provides both unmanaged and unmanaged APIs for audio capture and control, as described in the following sections.

Kinect SDK C++ Audio API

Unmanaged applications can capture the Kinect sensor's microphone array in several ways, as described in the following sections.

Raw Audio Capture

Applications can use the Windows Audio Session API (WASAPI) to capture the raw audio stream from the Kinect sensor's microphones. For an example of how to capture a raw audio stream, see the AudioCaptureRaw sample. For more information, see "AudioCaptureRaw Walkthrough" on the SDK Beta website.

KinectAudio DMO

Windows Vista® and later versions include a voice-capture digital signal processor (DSP) that supports microphone arrays. Developers typically access that DSP through a DMO, which is a standard COM object that can be incorporated into a Microsoft DirectShow graph or a Microsoft Media Foundation topology. The Kinect for Windows SDK Beta includes an extended version of the Windows microphone array DMO—referred to here as the MSRKinectAudio DMO—to support the Kinect microphone array.

Although the internal details are different, the KinectAudio DMO supports the same interfaces as the standard microphone array DMO and works in much the same way. However, the KinectAudio DMO:

- Has its own class identifier (CLSID), **CLSID_CMSRKinectAudio**.
- Supports an additional microphone mode, which is customized to support the Kinect microphone array.
- Includes beamforming and source localization functionality, which it exposes through an additional interface, **ISoundSourceLocalizer**.

The beamforming functionality supports 11 fixed beams, which range from −50 to +50 degrees in 10-degree increments. Applications can use the DMO's adaptive beamforming option—which automatically selects the optimal beam—or specify a particular beam. The DMO also includes a source localization algorithm, which estimates the source direction.

This SDK Beta includes the following two samples that show how to use the KinectAudio DMO to capture an audio stream from the Kinect sensor's microphone array:

- MicArrayEchoCancellation shows how to capture sound and determine the selected beam and source direction by using the DMO as a DirectShow source. For more information, see "MicArrayEchoCancellation Walkthrough" on the SDK Beta website.
- MFAudioFilter shows how to capture sound by using the KinectAudio DMO as a filter object in a Media Foundation topology. For more information, see "MFAudioFilter Walkthrough" on the SDK Beta website.

Kinect SDK Managed Audio API

Kinect for Windows SDK Beta includes a managed Audio API, which is basically a wrapper over the KinectAudio DMO that supports the same functionality but is much simpler to use. The managed API allows applications to configure the DMO and perform operations such as starting, capturing, and stopping the audio stream. The managed API also includes events that provide the source and beam directions to the application.

Important The managed Kinect Audio API runs the DMO on a background thread, which requires the multithreaded apartment (MTA) threading model. Otherwise, the interop layer throws an exception. If you use a programming model such as WPF that requires single-threaded apartment (STA) threading, use a separate MTA thread for Kinect Audio.

The RecordAudio sample shows how to use the managed API to capture sound and monitor the selected beam and source direction. For more information, see "RecordAudio Walkthrough" on the SDK Beta website.

How to Use Microsoft.Speech with the Kinect Microphone Array

Speech recognition is a key aspect of natural user interface and is supported in Windows by the Microsoft.Speech platform. The Kinect for Windows SDK provides the necessary infrastructure for managed applications to use the Kinect microphone with the Microsoft.Speech API, which supports the latest acoustical algorithms. This SDK Beta includes a custom acoustical model that is optimized for the Kinect sensor's microphone array.

Note MSDN provides limited documentation for the Microsoft.Speech API. You should instead use the HTML Help file (CHM) that is included with the Speech SDK, which can be found at Program Files\Microsoft Speech Platform SDK\Docs.

To use Microsoft.Speech with the Kinect sensor, you must install the following components:

- Microsoft Speech Platform - Software Development Kit, version 10.2 (x86 edition)
- Kinect for Windows Runtime Language Pack, version 0.9
  (acoustic model from Microsoft Speech Platform for the Kinect for Windows SDK Beta)
- Microsoft Speech Platform - Server Runtime, version 10.2 (x86 edition)

  The Kinect for Windows SDK Beta runtime is x86-only, so you must download the x86 version of the speech runtime for x86 or x64 systems.

PART 4—Resources

Tips and Troubleshooting

Troubleshooting Tips for Driver Installation

1. Make sure your Kinect device is plugged into an external power source, not just your PC.
2. If all elements of the composite devices do not appear in Device Manager:
    - Unplug the Kinect sensor, and reconnect it.
    - Make sure that the sensor is connected to a functioning USB port—for example, connect another USB device such as a mouse that you've already used on your computer.
    - Make sure that the sensor does not share the USB hub with any other devices.
    - Make sure to uninstall any previous driver, even if it's a driver installed with the Kinect for Windows SDK Beta. To do this:
        - Unplug the Kinect sensor.
        - Follow the instructions from the driver's manufacturer or select it in **Programs and Features** in Control Panel and click **Uninstall**. Also, check your system's registry to ensure than any previous driver has been removed, other than one provided with the SDK Beta.
        - Uninstall previous versions of this SDK Beta driver in **Programs and Features** in Control Panel. Then reinstall it from the original SDK Beta download package.
        - Plug in the Kinect sensor again.

General Kinect Device Troubleshooting

For issues that might be related to the Kinect sensor setup and sensors, see these Kinect topics on the Xbox Support web site:

- Kinect Setup—positioning the sensor and setting up the play space.
- More Topics—cleaning the sensor, mounting, and replacement parts.

Detecting the Absence of a Kinect Sensor

If no Kinect sensor is connected to the PC when the application tries to initialize image or audio capture:

- **NuiInitialize** returns HRESULT error 0x80080014.
- **Microsoft.Research.Kinect.Nui.Runtime.Initialize** throws a **System.InvalidOperationException**.
- The **Microsoft.Research.Kinect.Audio.KinectAudioSource** constructor throws a **System.InvalidOperationException**.

Audio

- When running the MicArrayEchoCancellation sample, you should be streaming data before you start the capture loop—for example, playing a music file on Windows Media Player.

DirectX Runtime Errors in Samples

- Make sure that the driver for your PC graphics card is up to date and is installed properly.
- Make sure that you have Version 9.0c or later of the DirectX runtime.

Troubleshooting Blue Screens and Deadlocking

- Make sure that webcams and other high-bandwidth USB devices connected to a different USB bus than the Kinect sensor.
- Make sure your Kinect device is connected to your PC when you launch a debugging session.
- If you unplug the Kinect sensor while streaming is underway or if some other unexpected error occurs during streaming, all subsequent **NuiXxx** calls except **NuiShutdown** fail and return the error code of the streaming error. The application must call **NuiShutdown** and then **NuiInitialize**—or the managed interface equivalents—before it can use the sensor again.

More SDK Beta Troubleshooting Tips

For additional notes, see these resources on the SDK Beta website:

- Readme for Kinect for Windows SDK Beta
- FAQ for Kinect for Windows SDK Beta

Terminology for Kinect and NUI

This section contains terminology used in Kinect for the Windows SDK Beta from Microsoft Research.

beamforming algorithm

  An algorithm that determines the direction of the sound source in the horizontal plane.

CDRP

  Color, depth, and pixel registration.

depth stream

  The data that is produced by the depth image camera in the Kinect sensor. Each frame in the stream contains the distance, in millimeters, to the nearest object at a particular x and y coordinate in the depth sensor's field of view. Two depth data streams are available, one with a frame size of 320×240 pixels and another with a frame size of 80×60 pixels.

Kinect sensor

  A physical device that contains cameras and a microphone array. The sensor connects to a PC by a USB cable, although Windows Plug and Play does not identify it as a USB device.

latency

  The required time for isochronous data to travel from the USB isochronous device to the client application. Many factors can affect latency, such as the video frame format (uncompressed, compressed, and so on), the size of the video frame, the data rate, and the CPU speed.

natural user interface (NUI)

  An evolving model for human-computer interaction that is context-appropriate and adaptive. Because the NUI exploits a user's existing skills and expectations, it is easy to learn. A NUI might incorporate speech, gesture, touch, or location, depending upon the application and the user's environment.

USB isochronous interface

  An interface that supports four USB data flow types: isochronous, control, interrupt, and bulk. The Kinect sensor has an isochronous interface, which is typically used for audio, video, and other

streaming data from cameras and similar devices. The interface provides constant bandwidth that is reserved exclusively for a particular device.

YUV stream

A stream that contains digital video data in which the three components are Y' (luma, which represents brightness) and two chroma values (Cb and Cr), which measure color difference. Because the human eye is more sensitive to differences in brightness than in hue, the chroma values are sampled at half the rate of the luma values, so a YUV stream contains two Y' values for each Cb and Cr. Consequently, at 8 bits per pixel, YUV data is more compressed than RGB data because only 4 bytes are needed for each two pixels. YUV color provides 16-bit, gamma-corrected linear UYVY-formatted color bitmaps. For more information, see About YUV Video in the MSDN library.

References

This section provides links to additional information about the Kinect for Windows SDK Beta from Microsoft Research and related topics. See also FAQ for Kinect for Windows SDK Beta.

Microphone Arrays in Windows

About WASAPI (Windows Audio Session API)

Beamforming

How to Build and Use Microphone Arrays for Windows Vista

Microphone Array Support in Windows Vista

NUI and Kinect

"CES 2010: NUI with Bill Buxton" on Channel 9
Real-Time Human Pose Recognition in Parts from a Single Depth Image

SDKs and Tools from Microsoft

DirectX SDK – June 2010 version

Kinect for Windows Runtime Language Pack, version 0.9
(acoustic model from Microsoft Speech Platform for the Kinect for Windows SDK Beta)

Microsoft .NET Framework 4.0

Microsoft Speech Platform - Server Runtime, version 10.2 (x86 edition)

Microsoft Speech Platform - Software Development Kit, version 10.2 (x86 edition)

Microsoft Visual C++ 2010 Express

Microsoft Visual C# 2010 Express

Microsoft Visual Studio 2010 Professional—trial edition

APPENDIX

How to Build an Application with Visual Studio Express

To work with the samples in the Kinect for Windows SDK Beta and to develop your own applications with this SDK Beta, you can use Visual Studio C++ 2010 Express or any other version of Visual Studio 2010. Visual Studio C++ Express is available at no cost.

To install Visual Studio 2010 Express

1. From the following web page, install Visual Studio 2010 Express for either C++ or C#:

   http://www.microsoft.com/express/downloads/

2. After the installation completes, check for updates on Windows Update. Install any relevant updates and reboot your computer if you are prompted to do so.

   Tip It's a good idea to keep all the software on your computer up to date.

3. The first time you start Visual Studio Express, the program prompts you for a registration code.

   To acquire the code, follow the instructions on the screen or see the following website: http://www.microsoft.com/express/support/regins/

   You need a Windows Live ID to register. If you use Messenger, Hotmail, or Xbox Live, you already have a Windows Live ID. If not, you can sign up at https://signup.live.com/signup.aspx?lic=1

Getting Started with Visual Studio Express

Use the following links to review introductory information on MSDN:

- Visual C++ Express Edition:

  Getting Started with Visual C++ Express Edition

  Getting Started with C++ and Visual C++ Express Edition

  For help with common Visual C++ tasks, see "How Do I?" Videos in the Visual C++ Developer Center.

- Visual C# Express Edition:

  Getting Started with Visual C#

  Using the Visual C# Development Environment

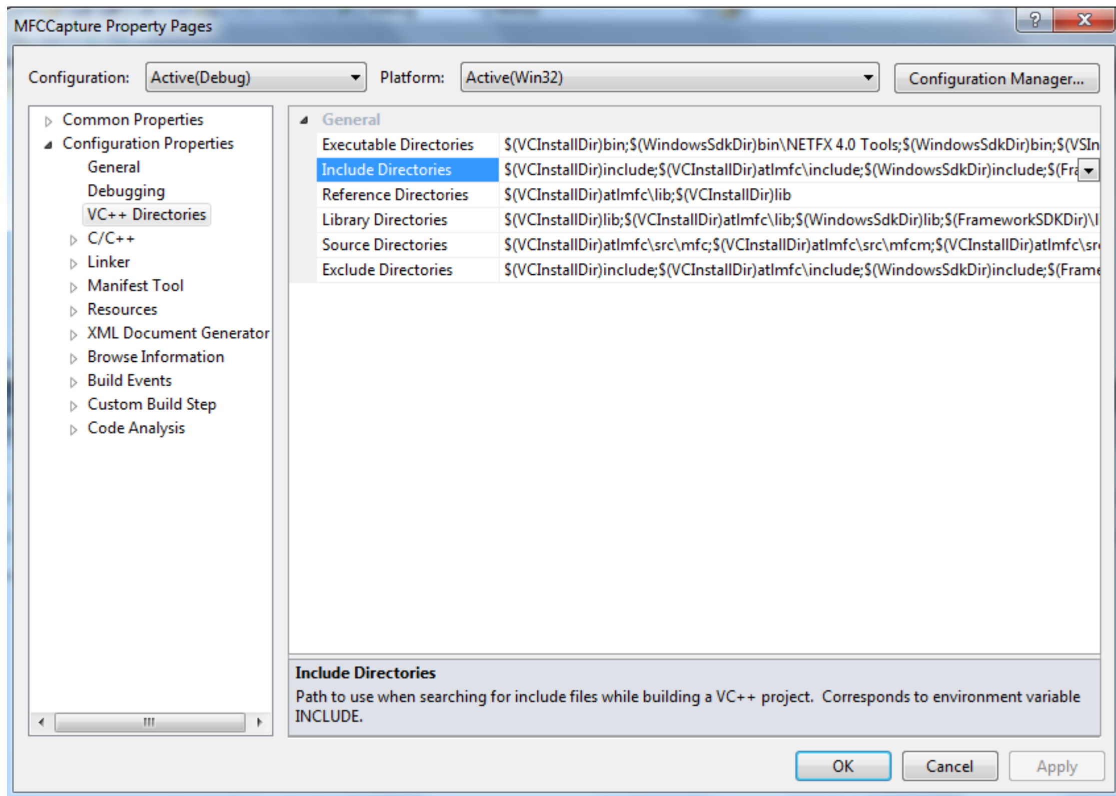
  For help with common Visual C# tasks, see "How Do I?" Videos in the Visual C# Developer Center.

Configuring the Development Environment

Visual Studio Express Edition for either C++ or C# includes an integrated development environment (IDE) in which you can code, debug, and test your applications. If you install Visual Studio Express in the default location and accept its defaults for project files, the default locations for source, header, and library files will also be correct. However, in some cases you might need to add folders to the defaults.

To change default file locations

1. Start your version of Microsoft Visual Studio Express from the **Start** menu.
2. Click **File**, click **Open**, and then click **Project/Solution**. Navigate to the folder that contains the solution file for the sample or application.
3. If the **Project** menu is not visible, press **F7** to try to build the solution. When you build the solution, Visual Studio Express displays the **Project** menu.
3. On the **Project** menu, click **Properties**.
4. In the left column of the property page, click **Configuration Properties** and then **VC++ Directories** (or **VC# Directories**, depending on your installation) to see a list of the locations that the development environment searches for files.
5. Click the item to change, and then click the drop-down arrow on the far right to edit the locations.



Creating Projects in Visual Studio

In Visual Studio, you organize development work into projects and solutions.

To create your first project in Visual Studio

1. Follow the steps for How to: Create Solutions and Projects on MSDN.
2. To add files to your project, follow the steps in How to: Add and Remove Solution Items on MSDN.

Note The walkthroughs for the Kinect for Windows SDK Beta describe the steps to use full Visual Studio, rather than Visual Studio Express. The development environments are similar, but some menus are organized differently because full Visual Studio provides more extensive capabilities. In particular, Visual Studio Express does not have a **Build** menu. You can find the **Build Solution** command on the **Debug** menu.

To build and test your project

1. Follow the steps outlined in Building in Visual Studio on MSDN.

   See also Walkthrough: Building a Project (C++).

2. To start using the debugger, see Building and Debugging on MSDN.

   See also Walkthrough: Testing a Project (C++).