The ActionXL Motion Controller Software Development Kit

The ActionXL Motion Controller software development kit is the fastest, easiest way to add motion sensing to your games and take advantage of the hottest trend in gaming today.

Windows sees the ActionXL Motion-Controller as a standard 3-axis, 2-button game controller. Therefore, any joystick calls from a game will be answered by Windows, with whatever game controllers Windows has identified. Windows identifies the ActionXL controller as “Motion Controller”. If your game is written to take advantage of joystick inputs, you already have all of the components you need to take advantage of ActionXL’s Motion Controller.

To address the joystick, refer to Microsoft’s DirectInput Tutorial 3: Using the Joystick. This tutorial shows you how to enumerate a joystick attached to a system and set it up for input using the Microsoft DirectInput system.

The first step in this tutorial is to enumerate devices; that is, to see what joysticks are available. As part of this process you initialize each joystick device and set its desired characteristics.

You then use the IDirectInputDevice8 Interface methods to retrieve data from each joystick.

Sample Code for the Motion Controller is provided under the heading Sample Motion Controller Control Class.

Note: DirectInput will eventually be replaced by DirectX. For a guide to using DirectX please refer to Microsoft’s DirectX SDK chapter covering DirectX Input.

Included Files

1. Binary DLL with Visual Studio Intellisense XML file.
2. Source code (in C#) for MotionControllerManager and MotionDetector
3. Readme.txt

Note: *Managed DirectX and DirectInput necessary to run manager (also requires .NET framework)*

Motion Controller Manager
This library enables the easy detection, enumeration, and reading of Motion Controllers attached to the system. It also includes a detector that enables the easy detection of controller orientation and some simple gestures.
Please note: Requires Managed DirectInput and DirectX to run.

Usage: (Also available in the readme file)

Instantiate the MotionControllerManager class. Call ***MotionControllerManager.initialize***, with an argument of true to automatically initialize a motion detector (the orientation and simple gesture detector), or false if you do not want to use the detector or plan to initialize it later. A ***NoMotionControllerFoundException*** will be thrown if no Motion Controllers are attached and ***MotionControllerManager.initialize*** is called.

Call ***MotionControllerManager.update*** to poll each Motion Controller. The argument to that function should be the number of seconds since it was last called.

MotionControllerManager.getButton returns true if the zero-indexed button specified is down on the zero-indexed controller specified (***MotionControllerManager.getButton***(0,0) returns the status of the left button on the first controller).

MotionControllerManager.getButtonPress returns true if the specified button on the specified controller was pressed between the most recent call to ***MotionControllerManager.update*** and the previous one.

MotionControllerManager.getAxis returns the value of the specified axis (0 for X, 1 for Y, and 2 for Z) from -2048 to +2047. Note that the extreme values represent 2Gs of acceleration, and therefore cannot be attained by tilt alone. Tilting will only get about 40% of the way to the maximum.

MotionControllerManager.getDetector returns the detector attached to the specified joystick, which can then be used to read orientation or test for some simple gestures.
MotionControllerManager.getDetector throws a ***DetectorsNotInitializedException*** if the detectors have not been initialized.

Creating DirectInput Device Objects
There are many ways to find out what input devices are currently attached to the user's computer. More often than not, you will have to capture input from more than one device.

The following C# example code illustrates how to create a Device object for a joystick.

[C#]
```
private void InitDevices()
{
    //create joystick device.
   foreach(
      DeviceInstance di in
      Manager.GetDevices(
         DeviceClass.GameControl,
         EnumDevicesFlags.AttachedOnly))
   {
      joystick = new Device(di.InstanceGuid);
      break;
   }

   if(joystick == null)
   {
      //Throw exception if joystick not found.
      throw new Exception("No joystick found.");
   }
}
```

Configuring DirectInput Device Objects

Because the input devices you capture from are a shared system resource, you have to configure them to cooperate with other system resources. You do this by calling the SetCooperativeLevel method of the Device objects you created with a combination of flags from the CooperativeLevelFlags enumeration.
The following table describes CooperativeLevelFlags that can be used when setting the cooperative level.

Flag	Description
CooperativeLevelFlags.Background	This device can be used in the background and can be acquired at any time, even if the application window is not active.
CooperativeLevelFlags.Foreground	This device can only be used when the application window is in the foreground. When the application window loses focus, the device will no linger be acquired.
CooperativeLevelFlags.Exclusive	This device requires exclusive access. No other application can acquire this device. For security reasons, this flag cannot be used in conjunction with the background flag on certain devices such as a keyboard and mouse.
CooperativeLevelFlags.NonExclusive	This device can be shared with any other application that does not require exclusive access.
CooperativeLevelFlags.NoWindowsKey	This flag disables the window's key.

Another configuration task is setting the range of a joystick axis. This allows you to set a numeric range to the complete distance traveled on a joystick axis. Even if the joystick you are using does not support the resolution of the range you assign, DirectInput will translate the joystick's resolution into the range you assign.

The following C# example code demonstrates how to configure the devices created in the previous sample.

[C#]

```
//Set mouse axis mode absolute.
mouse.Properties.AxisModeAbsolute = true;

//Set joystick axis ranges.
foreach(DeviceObjectInstance doi in joystick.Objects)
{
   if((doi.ObjectId & (int)DeviceObjectTypeFlags.Axis) != 0)
   {
      joystick.Properties.SetRange(
         ParameterHow.ById,
         doi.ObjectId,
         new InputRange(-5000,5000));
   }
}

//Set joystick axis mode absolute.
joystick.Properties.AxisModeAbsolute = true;

//set cooperative level.
keyboard.SetCooperativeLevel(
   this,
   CooperativeLevelFlags.NonExclusive |
   CooperativeLevelFlags.Background);

mouse.SetCooperativeLevel(
```

```
    this,
    CooperativeLevelFlags.NonExclusive |
    CooperativeLevelFlags.Background);

joystick.SetCooperativeLevel(
    this,
    CooperativeLevelFlags.NonExclusive |
    CooperativeLevelFlags.Background);

//Acquire devices for capturing.
keyboard.Acquire();
mouse.Acquire();
joystick.Acquire();
```

Capturing from DirectInput Device Objects

Normally you would want to capture the device object's state at a certain point of your application.
The C# example code below contains methods for capturing device input from the devices created in the previous examples, and updating a label control. They can be called by an application whenever you need to capture DirectInput device input.

[C#]

```
private void UpdateKeyboard()
{
    string info = "Keyboard: ";

    //Capture pressed keys.
    foreach(Key k in keyboard.GetPressedKeys())
    {
        info += k.ToString() + " ";
    }
    lbKeyboard.Text = info;
}

private void UpdateMouse()
{
    string info = "Mouse: ";

    //Get Mouse State.
    MouseState state = mouse.CurrentMouseState;

    //Capture Position.
    info += "X:" + state.X + " ";
    info += "Y:" + state.Y + " ";
    info += "Z:" + state.Z + " ";

    //Capture Buttons.
    byte[] buttons = state.GetMouseButtons();
    for(int i = 0; i < buttons.Length; i++)
    {
        if(buttons[i] != 0)
        {
            info += "Button:" + i + " ";
        }
    }

    lbMouse.Text = info;
}
```

```
private void UpdateJoystick()
{
    string info = "Joystick: ";

    //Get Mouse State.
    JoystickState state = joystick.CurrentJoystickState;

    //Capture Position.
    info += "X:" + state.X + " ";
    info += "Y:" + state.Y + " ";
    info += "Z:" + state.Z + " ";

    //Capture Buttons.
    byte[] buttons = state.GetButtons();
    for(int i = 0; i < buttons.Length; i++)
    {
        if(buttons[i] != 0)
        {
            info += "Button:" + i + " ";
        }
    }

    lbJoystick.Text = info;
}
```

Sample Motion Controller Control Class (example code)

```
using System;
using System.Collections.Generic;
using Microsoft.DirectX.DirectInput;


/*  Motion Controller Control Class
    *      * This class enables the easy detection, enumeration, and reading of Motion Controllers attached to the
system.
    * It requires Managed DirectInput and DirectX to run.
    *
    *
    * (c) ActionXL 2007
    * Written by Benjamin Kalb
    */


  class MotionControllerManager
  {
      List<Device> applicationDevices;
      List<JoystickState> currentStates, lastStates;

      /// <summary>
      /// The number of attached and detected Motion Controllers
      /// </summary>
      public int Count
      {
          get { return applicationDevices.Count; }
      }

      public MotionControllerManager()
      {
```

```
        applicationDevices = new List<Device>();
        currentStates = new List<JoystickState>();
        lastStates = new List<JoystickState>();
    }

    /// <summary>
    /// Clear all currently detected joysticks and re-detect all joysticks attached to the system,
    /// enumerating all attached Motion Controllers.
    /// </summary>
    public void initialize()
    {
        //Reset stored Joysticks
        applicationDevices.Clear();
        currentStates.Clear();
        lastStates.Clear();
        //Initialize DirectInput
        // Enumerate Joysticks in the system.
        foreach (DeviceInstance instance in Manager.GetDevices(DeviceClass.GameControl,
EnumDevicesFlags.AttachedOnly))
        {
            // Create the device.  Accept only Motion Controllers
            if (instance.InstanceName == "Motion Controller")
            {
                applicationDevices.Add(new Device(instance.InstanceGuid));
                currentStates.Add(new JoystickState());
                lastStates.Add(new JoystickState());
            }
        }

        if (applicationDevices.Count == 0)
        {
            //fail
            throw new NoMotionControllerFoundException("No Motion Controller Found");
        }

        //loop through all detected controllers and set axis parameters.
        foreach (Device applicationDevice in applicationDevices)
        {
            // Set the data format to the c_dfDIJoystick pre-defined format.
            applicationDevice.SetDataFormat(DeviceDataFormat.Joystick);

            // Enumerate all the objects on the device.
            foreach (DeviceObjectInstance d in applicationDevice.Objects)
            {
                // For axes that are returned, set the DIPROP_RANGE property for the
                // enumerated axis in order to scale min/max values.
                if ((0 != (d.ObjectId & (int)DeviceObjectTypeFlags.Axis)))
                {
                    // Set the range for the axis.  The Motion Controller has 12 bits of precision (0-4095),
                    //so we use that range and center it about zero.
                    applicationDevice.Properties.SetRange(ParameterHow.ById, d.ObjectId, new InputRange(-2048,
+2047));
                }
            }
        }

    }
```

```
    /// <summary>
    /// Poll each motion controller and get its state.  This function  should be called every frame.
    /// </summary>
    public void update()
    {
        if (applicationDevices.Count == 0)
            return;

        //set the last states (to check for button presses, etc.)
        for (int i = 0; i < currentStates.Count; i++)
        {
            lastStates[i] = currentStates[i];
        }

        //Loop through all the Motion Controllers
        foreach (Device applicationDevice in applicationDevices)
        {
            try
            {
                // Poll the device for info.
                applicationDevice.Poll();
            }
            catch (InputException inputex)
            {
                if ((inputex is NotAcquiredException) || (inputex is InputLostException))
                {
                    // Check to see if either the app
                    // needs to acquire the device, or
                    // if the app lost the device to another
                    // process.
                    try
                    {
                        // Acquire the device.
                        applicationDevice.Acquire();
                    }
                    catch (InputException)
                    {
                        // Failed to acquire the device.
                        // This could be because the app
                        // doesn't have focus.
                        return;
                    }
                }

            } //catch(InputException inputex)

            // Get the state of the device.
            try { currentStates[applicationDevices.IndexOf(applicationDevice)] =
applicationDevice.CurrentJoystickState; }
            // Catch any exceptions. None will be handled here,
            // any device re-aquisition will be handled above.
              catch (InputException)
            {
                return;
            }
        }
    }
```

```
/// <summary>
/// Get the value of an axis on Motion Controller 0
/// </summary>
/// <param name="axis">The axis to check (X=0, Y=1, Z=2)</param>
public int getAxis(int axis)
{
   return getAxis(axis, 0);
}

/// <summary>
/// Get the value of an axis on a Motion Controller
/// </summary>
/// <param name="axis">The axis to check (X=0, Y=1, Z=2)</param>
/// <param name="joyStick">The zero-based index of the motion controller to read</param>
public int getAxis(int axis, int joyStick)
{
   if (joyStick >= applicationDevices.Count)
   {
      return 0;
   }
   switch (axis)
   {
      case 0:
         return currentStates[joyStick].X;
      case 1:
         return currentStates[joyStick].Y;
      case 2:
         return currentStates[joyStick].Z;
      default:
         return 0;
   }
}

/// <summary>
/// Check the status of a button on Motion Controller 0
/// </summary>
/// <param name="button">The zero-based index of the button to read</param>
/// <returns>True if the button is down</returns>
public bool getButton(int button)
{
   return getButton(button, 0);
}

/// <summary>
/// Check the status of a button on a Motion Controller
/// </summary>
/// <param name="button">The zero-based index of the button to read</param>
/// <param name="joyStick">The zero-based index of the motion controller to read</param>
/// <returns>True if the button is down</returns>
public bool getButton(int button, int joyStick)
{
   if (joyStick >= applicationDevices.Count)
   {
      return false;
   }
   return currentStates[joyStick].GetButtons()[button] > 0;
}
```

```
        /// <summary>
        /// Check whether a button was pressed this frame (getButton() was false last update() and is true this
update()) on Motion Controller 0
        /// </summary>
        /// <param name="button">The zero-based index of the button to read</param>
        /// <returns>True if the button was just pressed</returns>
        public bool getButtonPress(int button)
        {
            return getButtonPress(button, 0);
        }

        /// <summary>
        /// Check whether a button was pressed this frame (getButton() was false last update() and is true this
update()) on a Motion Controller
        /// </summary>
        /// <param name="button">The zero-based index of the button to read</param>
        /// <param name="joyStick">The zero-based index of the motion controller to read</param>
        /// <returns>True if the button was just pressed</returns>
        public bool getButtonPress(int button, int joyStick)
        {
            if (joyStick >= applicationDevices.Count)
            {
                return false;
            }
            return currentStates[joyStick].GetButtons()[button] > 0 && lastStates[joyStick].GetButtons()[button] == 0;
        }
    }

    class NoMotionControllerFoundException : Exception
    {
        public NoMotionControllerFoundException(string message)
            : base(message)
        {
        }
    }
```

Technical Articles

a. Tilt Sensing With Kionix Accelerometers
b. Position determination using Accelerometers
c. Microsoft Introduction to DirectInput
d. Microsoft DirectX SDK