Xin Version 1.0

Xin, short for XNA input manager (X and in), is a free open source library for handling input in your XNA games. Xin is distributed under the GNU GPL for open source software. That means that you are free to use and change Xin as long as you respect the authors rights. It would be nice if you put a credit to me in your game, or link to me on your web site, if you found Xin useful. You can make a donation if you find Xin useful on my web site: http://www.xnagpa.net/.

Right now there are four versions of Xin. One is for the Xbox 360 and the other is for Windows for both XNA Game Studio 3.0 and XNA Game Studio 3.1. A partner is creating versions for the Zune and the Zune HD and when they are ready they will be released as well. With the release of Visual Studio 2010 and the upcoming release of XNA Game Studio 4.0 there will be a new release of Xin for XNA 4.0 as well when that time comes.

Xin was written using a game component and it very easy to add to your games.

- Add the appropriate XInputManager library to your solution
- Add a reference to the library to any projects in the solution that you want to use Xin in
- Add a using statement for the XInputManager namespace
- In the constructor of your game create an instance of Xin and add it to the list of game components for your game

Here is the sample code of the Game1 class that is created when you create a new XNA project.

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

// using statement for the XInputManager namespace
using XInputManager;

namespace XinTestGame
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        // Field used for the Xin Game Component
        Xin xinComponent;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
```

```
            // Create an instance of the Xin Game Component and add
            // it to the list of game components.
            xinComponent = new Xin(this);
            Components.Add(xinComponent);
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content.  Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here
        }

        /// <summary>
        /// UnloadContent will be called once per game and is the place to unload
        /// all content.
        /// </summary>
        protected override void UnloadContent()
        {
            // TODO: Unload any non ContentManager content here
        }

        /// <summary>
        /// Allows the game to run logic such as updating the world,
        /// checking for collisions, gathering input, and playing audio.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Update(GameTime gameTime)
        {
            // Allows the game to exit if the back button on game pad one is down
            if (Xin.IsButtonDown(Buttons.Back))
                this.Exit();

            // Allows the game to exit by checking if the exit key was pressed once
            if (Xin.CheckKeyPress(Keys.Escape))
                this.Exit();

            base.Update(gameTime);
        }

        /// <summary>
        /// This is called when the game should draw itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            // TODO: Add your drawing code here

            base.Draw(gameTime);
        }
    }
}
```

Xin has many methods and properties for you to use in your game and many are named after properties and methods you are already used to. They are all static so you reference them using the class name, Xin. There are two versions of each property and method. The first version refers to the current input state of your game. The second version refers to the input state of your game in the last frame of the game. They last state properties and methods are added so you can check for single presses of keys, mouse buttons, game pad buttons, etc. and you will find them useful for other reasons as well.

Here is a run down of the public properties/methods for each input device:

Keyboard

- **KeyboardState Xin.KeyboardState**
  Property that returns the current state of the keyboard.

- **KeyboardState Xin.LastKeyboardState**
  Property that returns the state of the keyboard in the last frame of the game.

- **bool Xin.CheckKeyPress(Keys key)**
  Method that checks for a single press of the key specified by checking **key** was down in the last frame and up in the current frame. Returns **true** if the key was pressed exactly once and **false** otherwise.

- **bool Xin.IsKeyDown(Keys key)**
  Method that checks if the key specified by **key** is currently down. Returns **true** if the key is down and **false** otherwise.

- **bool Xin.IsKeyUp(Keys key)**
  Method that checks if the key specified by **key** is currently up. Returns **true** if the key is up and **false** otherwise.

- **bool Xin.LastIsKeyDown(Keys key)**
  Method that checks if the key specified by **key** was down in the last frame of the game.. Returns **true** if the key was down and **false** otherwise.

- **bool Xin.LastIsKeyDown(Keys key)**
  Method that checks if the key specified by key was up in the last frame of the game.. Returns **true** if the key is down and **false** otherwise.

- **Keys[] Xin.GetPressedKeys()**
  Method that returns an array of **Keys** that are currently down.

- **Keys[] Xin.LastPressedKeys()**
  Method that returns an array of **Keys** that were down in the last frame of the game.

Mouse

(Only available in Windows games)

For the mouse there is an enumeration called **MouseButtons** that allows you to easily select the mouse button that you wish to use. They include: { **Left**, **Middle**, **Right**, **X1**, **X2** } **X1** and **X2** are for the two possible extra buttons on the mouse.

- **MouseState Xin.MouseState**
  Property that returns the current state of the mouse.

- **MouseState Xin.LastMouseState**
  Property the returns the state of the mouse in the last frame of the game.

- **Point Xin.MouseAsPoint**
  Property that returns a **Point** that represents the location of the mouse relative to the upper left corner of the client area.

- **Vector2 Xin.MouseAsVector2**
  Property that returns a **Vector2** that represents the location of the mouse relative to the upper left corner of the client area.

- **Point Xin.LastMouseAsPoint**
  Property that returns a **Point** that represents the location of the mouse relative to the upper left corner of the client area in the last frame of the game.

- **Vector2 Xin.LastMouseAsVector2**
  Property that returns a **Vector2** that represents the location of the mouse relative to the upper left corner of the client area client are in the last frame of the game.

- **bool CheckMousePress(MouseButton button)**
  Method that checks to see if the button on the mouse specified by **button** was clicked exactly once by checking the button was down in the last frame of the game and up in the current frame of the game. This method calls private methods in Xin that detect clicks using the **MouseState** class. Returns **true** if the button was clicked once and **false** otherwise.

- **bool IsMouseDown(MouseButton button)**
  Method that checks to see if the mouse button specified by **button** is currently down. Returns **true** if the button is currently down and **false** otherwise.

- **bool IsMouseUp(MouseButton button)**
  Method that checks to see if the mouse button specified by **button** is current up. Returns **true** if the button is current up and **false** if it is no.

- **bool IsLastMouseDown(MouseButton button)**
  Method that checks to see if the mouse button specified by **button** is currently down. Returns **true** if the button was down in the last frame of the game and **false** otherwise.

- **bool IsLastMouseUp(MouseButton button)**

Method that checks to see if the mouse button specified by **button** was up in the last frame. Returns **true** if the button was up in the last from of the game and **false** otherwise.

Xbox 360 Controller
(Game Pad)

Unlike with the mouse and keyboard there can be more than 1 game pad. To choose the game pad you are interested in you use an enumeration called **PlayerIndex** as a parameter to the methods. There are also overloads of the methods that do not take a **PlayerIndex** parameter that return values according to the controller referenced by **PlayerIndex.One** for easier access to the controller.

- **GamePadState GamePadState(PlayerIndex index)**
  Method that gets the current state of the game pad referenced by **index**.
- **GamePadState GamePadState()**
  Overload that returns the state of the game pad for **PlayerIndex.One**.

- **GamePadState LastGamePadState(PlayerIndex index)**
  Method that gets the state of the game pad referenced by **index**.
- **GamePadState LastGamePadState()**
  Overload that returns the state of the game pad for **PlayerIndex.One**.

- **bool IsConnected(PlayerIndex index)**
  Method that returns if the game pad referenced by **index** is currently connected. Returns **true** if the game pad is connected and **false** otherwise. Normally you should only us the method **IsConnected(playerIndex)** after a call to **GamePad.GetState(playerIndex)** but Xin takes care of this for you automatically because it is a game component that updates itself.
- **bool IsConnected()**
  Overload that returns if game pad **PlayerIndex.One** is connected.

- **GamePadButtons GamePadButtons(PlayerIndex index)**
  Method that gets the state of the buttons on the game pad referenced by **index**.
- **GamePadButtons GamePadButtons()**
  Overload that returns the state of the buttons on the game pad for **PlayerIndex.One**.

- **GamePadButtons LastGamePadButtons(PlayerIndex index)**
  Method that gets the state of the buttons on the game pad referenced by **index** in the last frame of the game.
- **GamePadButtons LastGamePadButtons()**
  Overload that returns the state of the buttons on the game pad for **PlayerIndex.One** in the last frame of the game.

- **void Vibrate(PlayerIndex index, float leftMotor, float rightMotor)**
  Method for setting the vibration speed of the motors in the game pad referenced by **index**. The **leftMotor** and **rightMotor** values are between 0.0f and 1.0f.
- **void Vibrate(float leftMotor, float rightMotor)**
  Overload of the method that vibrates the motors for **PlayerIndex.One**.

- **void StopVibration(PlayerIndex index)**
  Method that stops the motors in the game pad referenced by **index**.
- **void StopVibration()**
  Overload that stops the vibration of the motors in game pad **PlayerIndex.One**.

- **Vector2 LeftThumb(PlayerIndex index)**
  Method that returns the state of the left thumb stick of the game pad referenced by **index** as a **Vector2**.
- **Vector2 LeftThumb()**
  Overload that returns the state of the left thumb stick for **PlayerIndex.One** as a **Vector2**.

- **Vector2 RightThumb(PlayerIndex index)**
  Method that returns the state of the right thumb stick of the game pad referenced by **index** as a **Vector2**.
- **Vector2 RightThumb()**
  Overload that returns the state of the right thumb stick for **PlayerIndex.One** as a **Vector2**.

- **Vector2 LastLeftThumb(PlayerIndex index)**
  Method that returns the state of the left thumb stick of the game pad referenced by **index** as a **Vector2** in the last frame of the game.
- **Vector2 LastLeftThumb()**
  Overload that returns the state of the left thumb stick for **PlayerIndex.One** as a **Vector2** in the last frame of the game.

- **Vector2 LastRightThumb(PlayerIndex index)**
  Method that returns the state of the right thumb stick of the game pad referenced by **index** as a **Vector2** in the last frame of the game.
- **Vector2 LastRightThumb()**
  Overload that returns the state of the right thumb stick for **PlayerIndex.One** as a **Vector2** in the last frame of the game.

- **GamePadDPad DPad(PlayerIndex index)**
  Method that returns the state of the direction pad of the game pad referenced by **index** as **GamePadDPad**.
- **GamePadDPad DPad()**
  Overload that returns the state of the direction pad for the game pad referenced by **PlayerIndex.One**.

- **GamePadDPad LastDPad(PlayerIndex index)**
  Method that returns the state of the direction pad of the game pad referenced by **index** as **GamePadDPad** in the last frame of the game.
- **GamePadDPad LastDPad()**

Overload that returns the state of the direction pad for the game pad referenced by **PlayerIndex.One** in the last frame of the game.

- **GamePadTriggers Triggers(PlayerIndex index)**
  Method that returns the state of the triggers of the game pad referenced by **index** as **GamePadTriggers**.
- **GamePadTriggers Triggers()**
  Overload of the method that returns the state of the triggers of the game pad referenced by **PlayerIndex.One**.

- **GamePadTriggers LastTriggers(PlayerIndex index)**
  Method that returns the state of the triggers of the game pad referenced by **index** as **GamePadTriggers** in the last frame of the game.
- **GamePadTriggers LastTriggers()**
  Overload of the method that returns the state of the triggers of the game pad referenced by **PlayerIndex.One** in the last frame of the game.

- **float LeftTrigger(PlayerIndex index)**
  Method that returns a value between 0.0f and 1.0f for the current state of the left trigger of the game pad referenced by **index**.
- **float LeftTrigger()**
  Overload of the method that returns the state of the left trigger for the game pad referenced by **PlayerIndex.One**.

- **float LastLeftTrigger(PlayerIndex index)**
  Method that returns a value between 0.0f and 1.0f for the state of the left trigger in the last frame of the game for the game pad referenced by **index**.
- **float LastLeftTrigger()**
  Overload of the method that returns the state of the left trigger for the game pad referenced by **PlayerIndex.One** in the last frame of the game.

- **float RightTrigger(PlayerIndex index)**
  Method that returns a value between 0.0f and 1.0f for the current state of the right trigger of the game pad referenced by **index**.
- **float RightTrigger()**
  Overload of the method that returns the state of the right trigger for the game pad referenced by **PlayerIndex.One**.

- **float LastRightTrigger(PlayerIndex index)**
  Method that returns a value between 0.0f and 1.0f for the state of the right trigger in the last frame of the game for the game pad referenced by **index**.
- **float LastRightTrigger()**

Overload of the method that returns the state of the right trigger for the game pad referenced by **PlayerIndex.One** in the last frame of the game.