SharpMedia Input Library Design

© 2008 Žiga Osolin – SharpMedia project, document version: DRAFT

Table Of Contents

[About 2](#_Toc196651512)

[Goals 2](#_Toc196651513)

[Overview 2](#_Toc196651514)

[Driver 3](#_Toc196651515)

[InputDevice 3](#_Toc196651516)

[Action Mapping 3](#_Toc196651517)

# About

Input library provides input facilities for arbitrary input device. It also provides input mapping facilities (action mapping) and abstraction for shell integration. It is based on simple-to-write driver architecture.

# Goals

Goals of input are:

* Support for input devices, enumeration and simple initialization;
* Support for multiple device to single hardware device mapping;
* Configurable action mapping;
* Event generation with event pumps, with support for repeated and double clicks events.

# Overview

Input is based on driver arhitecture. **InputService** is constructed with driver part. It is capable of enumerating devices and their creation. There will always be only one driver device, but they may be more input API devices linked to the same driver device.

Input system provides some layers for even simpler event handling. **EventPump** can be used to pool device and generate events on state changes. **EventProcessor** is a helper that dispatches input events through subscribed delegates. Action mapping allows key-combinations, gestures, button clicks and more to trigger events.

# Driver

Driver is a simple implementation of interfaces. Each device must know how to retrieve data from it. Everything else is handled by higher layers.

The base interface is **IInput** which exposes methods for creating input device listeners. Device listeners should be unique and only one **IInput** is valid at one time. Before any device is requested, the input is initialized with OS window. The **IInput** must also be able to enumerate devices and their capabilities.

**IInputDevice** is disposable and has only one member that is used to retrieve its state. The member fills button states and axis positions.

# InputService

**InputService** is smart driver part wrapper. It allows to enumerate device capabilities and create devices. There may be multiple smart API devices bound to the same driver device listener. Before **InputService** is used, it must be initialized with OS window.

# InputDevice

**InputDevice** represents an input device. It can be synhronized (state updated) or asked to retrieve any button/axis state. When you stop using it, you must dispose it.

# EventPump

**EventPump** is a helper class that can be assigned input devices. It will constantly pool them and generate **InputEvent**s on change. Those events are stacked on event queue. You can then obtain those events through **Get()** method.

**EventPump** can also generate repeated events (for example, when key is pressed for long time). To enable this, you must subscribe a delegate that returns the time span that is required in order to fire such repeated event. N-clicks are also supported. To enable this, you must provide an delegate that returns maximum time span between such clicks. For n-clicks, normal events are still triggerred.

# EventProcessor

**EventProcessor** is based on **EventPump**. Instead of using Get() method and handling events all at once, processor allows delegate to be executed on *filtered* events. Processor does basic filtering, such as splitting keyboard, mouse and custom device events. It also splits events based on devices. Following events are supported:

* KeyDown, KeyUp
* MouseButtonDown, MouseButtonUp, MouseAxis
* CustomEvent
* AllEvents

# Action Mapping

**ActionMapping** class ...