Seed Draft

Configuring Game Input Devices with InputSprocket

For InputSprocket 1.7

Preliminary

October 20, 1999
Technical Publications

Introduction

IMPORTANT

This is a preliminary document. Although it has been reviewed for technical accuracy, it is not final. Apple Computer, Inc. is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. You can check <http://developer.apple.com/techpubs/macos8/SiteInfo/whatsnew.html> for information about updates to this and other developer documents. To receive notification of documentation updates, you can sign up for ADC's free Online Program and receive their weekly Apple Developer Connection News e-mail newsletter. (See <http://developer.apple.com/membership/index.html> for more details about the Online Program.) ▲

InputSprocket is a subset of Apple Game Sprockets that allows you to easily configure game-oriented input devices such as joysticks, and gamepads. Today's input devices may have directional pads, levers, point-of-view hats, wheels, and buttons, and a game my use several input devices. Such complexity becomes impossible to emulate through a mouse and keyboard. The InputSprocket architecture provides a simple solution, allowing you to to easily configure a wide range of input devices.

This document assumes you are familiar with programming Macintosh computers. If you are building a game, you may also want to consult other Game Sprocket documentation:

- *Manipulating Displays Using DrawSprocket*
- *Simplifying Networked Gaming Using NetSprocket*

- SoundSprocket documentation (forthcoming)

This document currently covers InputSprocket in the following chapters:

- Chapter 2, "InputSprocket Reference," contains a complete programming reference, documenting the functions, data types, and constants available with InputSprocket.
- Appendix A, "Document Version History," describes changes made from previous versions of InputSprocket documentation.

For additional information about creating games for the Macintosh, you should check the Apple Developer games Web site:

<http://developer.apple.com/games/>

C H A P T E R

InputSprocket Reference

Contents

InputSprocket Reference

This chapter describes the InputSprocket application programming interface (API) introduced with InputSprocket 1.7. This chapter contains the following sections:

Note
This document describes version 1.7 of InputSprocket. For a summary of changes and additions between versions 1.0 and 1.7, see Appendix A. ◆

Functions

The following sections describe the functions provided by InputSprocket:

Invoking and Configuring InputSprocket

This section describes the functions that you use to start up InputSprocket and handle autoconfiguration.

■ `ISpGetVersion` (page 10) returns the version of InputSprocket installed.

- `ISpStartup` (page 11) starts up InputSprocket and loads all the InputSprocket drivers.
- `ISpInit` (page 11) initializes the high-level InputSprocket layer and autoconfigures all the devices.
- `ISpConfigure` (page 13) uses the high-level InputSprocket layer to generate a modal window where the user can match device elements with the game's input requirements.
- `ISpStop` (page 14) stops the flow of data into the virtual elements that began with the `ISpInit` call.
- `ISpSuspend` (page 15) suspends InputSprocket.
- `ISpResume` (page 15) resumes InputSprocket.
- `ISpTickle` (page 16) allows InputSprocket to give up time to other InputSprocket drivers.
- `ISpShutdown` (page 16) shuts down InputSprocket and unloads all InputSprocket drivers.

ISpGetVersion

Returns the version of InputSprocket installed.

```
NumVersion ISpGetVersion  (void);
```

function result The version number of InputSprocket.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpStartup

Starts up InputSprocket and loads all the InputSprocket drivers.

```
OSStatus ISpStartup (void);
```

function result A result code. See “Result Codes” (page 93).

DISCUSSION

Your application should call this function when it is ready to use InputSprocket.

VERSION NOTES

Introduced with InputSprocket 1.1.

ISpInit

Initializes the high-level InputSprocket layer and autoconfigures all the devices.

```
OSStatus ISpInit (
                    UInt32 count,
                    ISpNeed *needs,
                    ISpElementReference *inReferences,
                    OSType appCreatorCode,
                    OSType subCreatorCode,
                    UInt32 flags,
                    short setListResourceID
                    UInt32 reserved);
```

`count`	The number of input requirements the game has. Each requirement is described in an `ISpNeed` structure.
`needs`	A pointer to an array of `ISpNeed` structures. The order of the need structures in the array is important because the input devices will try to fulfill input requirements beginning with the first need structure in the array. More important requirements should be put first—for example, “jump” before “look at map.”

`inReferences`	A pointer to an array of virtual elements identifying those elements that can meet your game's input requirements. The array will contain the number of element references specified in the `count` parameter. You can use all the usual calls to get events or poll these element references.
`appCreatorCode`	The creator code of the application.
`subCreatorCode`	The subcreator code. InputSprocket and device drivers use a union of the creator and subcreator codes to save and restore preferences. For every pair of codes the game's requirements for input should be identical; otherwise, there is an unknown result. The subcreator code gives you a domain in which to have multiple different preference settings within any given application. You can also use this parameter to set a version number.
`flags`	Leave as 0.
`setListResourceID`	A resource of type `kISpSetListResourceType`. See "The InputSprocket Set List Resource" (page 79)
`reserved`	Reserved. Set to 0.
function result	A result code. See "Result Codes" (page 93).

DISCUSSION

The `ISpInit` function takes the number of input requirements in the `count` parameter, a pointer to an array of need structures in the `needs` parameter, and a pointer to an array of virtual element references in the `inReferences` parameter. A virtual element reference does not correspond to any physical control on a physical device. Devices push data into a virtual element reference when they have data that corresponds to the input requirement that reference represents. DrawSprocket provides two functions for creating virtual elements—`ISpElement_NewVirtual` (page 17) and `ISpElement_NewVirtualFromNeeds` (page 18).

To determine which elements can meet the game's requirements for input, `ISpInit` goes down a list of system devices and asks each device in turn if it can meet any of the requirements listed in the `needs` array. On the list of system devices, which is created the first time InputSprocket is loaded, the keyboard appears last and the mouse next to last. Note that you should not assume anything about the order the devices appear in the system list. If certain types of input devices are unsuited to the game, you can deactivate them by calling the function `ISpDevices_DeactivateClass` (page 35).

As each device tries to fulfill the input requirements, it looks at the need structures in the order in which they appear in the array. If a particular requirement has already been fulfilled by a prior device and has the `kInputSprocketNoMultiConfig` flag set in the `ISpNeed` structure `flags` field, the device will ignore it. The device driver keeps track of how its actual device elements are matched to the virtual element references it creates— in other words, which elements will meet which input requirements.

For each device driver, InputSprocket stores its configuration information, using the codes passed in the `appCreatorCode` and `subCreatorCode` parameters to identify them for future use.

SPECIAL CONSIDERATIONS

Do not call during interrupt time.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpConfigure

Uses the high-level InputSprocket layer to generate a modal window where the user can match device elements with the game's input requirements.

```
OSStatus ISpConfigure  (ISpEventProcPtr inEventProcPtr);
```

`inEventProcPtr`
: A pointer to an application-supplied function for handling update events. See `MyEventProc` (page 45) for more information about implementing this function.

function result A result code. See "Result Codes" (page 93).

DISCUSSION

The `ISpConfigure` function allows the user to modify the autoconfiguration. You pass the function a pointer to an application-defined function for handling events. If an event happens that the game needs to deal with, the game can handle the event and return True. If it does not handle the event, the game returns False. When the `ISpConfigure` call returns, the reconfiguring process is completed and any changes are saved to disk.

Note that you must call calling the function `ISpInit` (page 11) before calling this function.

SPECIAL CONSIDERATIONS

Do not call during interrupt time.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpStop

Stops the flow of data into the virtual elements and disposes of elements allocated by the `ISpInit` call.

```
OSStatus ISpStop  (void);
```

function result A result code. See “Result Codes” (page 93).

DISCUSSION

The `ISpStop` function stops data switched from device driver callbacks from being pushed into virtual elements. Your application should call this function before calling the function `ISpShutdown` (page 16).

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpSuspend

Suspends InputSprocket.

```
OSStatus ISpSuspend  (void);
```

function result A result code. See “Result Codes” (page 93).

DISCUSSION

You must suspend InputSprocket whenever your application receives Mac OS suspend events (for example, when the application switches to the background). It also can be called any time you want to stop getting InputSprocket data.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpResume

Resumes InputSprocket.

```
OSStatus ISpResume  (void);
```

function result A result code. See “Result Codes” (page 93).

DISCUSSION

You can call `ISpResume` in response to a Mac OS resume event, or anytime you want to resume running InputSprocket after having suspended it.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpTickle

Allows InputSprocket to give up time to other InputSprocket drivers.

```
OSStatus ISpTickle (void);
```

function result A result code. See “Result Codes” (page 93).

DISCUSSION

If you have input devices that require active participation by software to process properly (such as speech recognition), you can call this function to give time to other InputSprocket devices. Note that some other drivers may also benefit from this call.

In general you should call `ISpTickle` at least several times per second—10 to 20 times is sufficient, and not more than 100.

VERSION NOTES

Introduced with InputSprocket 1.1.

ISpShutdown

Shuts down InputSprocket and unloads all InputSprocket drivers.

```
OSStatus ISpShutdown (void);
```

function result A result code. See “Result Codes” (page 93).

DISCUSSION

Your application must call this function before it quits.

VERSION NOTES

Introduced with InputSprocket 1.1.

Manipulating Elements

This section describes functions you can use to create and manipulate elements, both physical and virtual.

- `ISpElement_NewVirtual` (page 17) creates a single virtual element for an element with data of a certain size.
- `ISpElement_NewVirtualFromNeeds` (page 18) allocates virtual elements for all items in a need structure array.
- `ISpElement_DisposeVirtual` (page 19) disposes of virtual elements.
- `ISpElement_GetDevice` (page 20) finds out what device an element belongs to.
- `ISpElement_GetGroup` (page 20) finds out what group an element belongs to.
- `ISpElement_GetInfo` (page 21) obtains an element information structure for an element.
- `ISpElement_GetConfigurationInfo` (page 21) obtains configuration information for an element.

ISpElement_NewVirtual

Creates a single virtual element for an element with data of a certain size.

```
OSStatus ISpElement_NewVirtual (
                    UInt32 dataSize,
                    ISpElementReference *outElement;
                    UInt32 flags);
```

`dataSize`	The size of the data the element receives, in bytes.
`outElement`	A pointer to a variable of type `ISpElementReference`. On return, `outElement` references the new virtual element.
`flags`	If you want the virtual element allocated in temporary memory pass `kISpVirtualElementFlag_UseTempMem` in the `flags` parameter. See “Virtual Element Flag” (page 75).
function result	A result code. See “Result Codes” (page 93).

DISCUSSION

In most cases, it is much easier to allocate elements using the function `ISpElement_NewVirtualFromNeeds` (page 18).

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElement_NewVirtualFromNeeds

Allocates virtual elements for all items in a need structure array.

```
OSStatus ISpElement_NewVirtualFromNeeds (
                UInt32 count,
                ISpNeed *needs,
                ISpElementReference *outElements
                UInt32 flags);
```

`count`	The number of need structures for which to allocate virtual elements.
`needs`	A pointer to an array of need structures.
`outElements`	A pointer to a variable of type `ISpElementReference`. On return, `outElements` points to an array of newly-created element references.
`flags`	If you want the virtual elements allocated in temporary memory pass `kISpVirtualElementFlag_UseTempMem` in the `flags` parameter. See “Virtual Element Flag” (page 75).
function result	A result code. See “Result Codes” (page 93).

DISCUSSION

This function only works if you are using built-in element kinds, which are described in “Built-in Element Kinds” (page 61). For other element kinds, use the `ISpElement_NewVirtual` function to create virtual elements.

Note that you typically call `ISpInit` (page 11) after calling this function, passing much of the same information.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElement_DisposeVirtual

Disposes of virtual elements.

```
OSStatus ISpElement_DisposeVirtual (
                    UInt32 count,
                    ISpElementReference *inElements);
```

`count`	The number of elements to dispose.
`inElements`	A pointer to the array of element references you want to dispose.
function result	A result code. See “Result Codes” (page 93).

DISCUSSION

You must call the function `ISpStop` (page 14) before disposing of any elements that are receiving data.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElement_GetDevice

Finds out what device an element belongs to.

```
OSStatus ISpElement_GetDevice (
                    const ISpElementReference inElement,
                    ISpDeviceReference *outDevice);
```

`inElement`	A reference to the element whose device you want to get. You cannot pass a reference to a virtual element in this parameter.
`outDevice`	On return, a device reference.
function result	A result code. See “Result Codes” (page 93).

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElement_GetGroup

Finds out what group an element belongs to.

```
OSStatus ISpElement_GetGroup (
                    const ISpElementReference inElement,
                    UInt32 *outGroup);
```

`inElement`	A reference to the element whose group you want to get.
`outGroup`	On return, a group ID. If the specified element does not belong to a group, the function sets this parameter to 0.
function result	A result code. See “Result Codes” (page 93).

DISCUSSION

Groups are arbitrary developer-assigned categories for input elements. One typical use is to differentiate between elements controlled by different players.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElement_GetInfo

Obtains an element information structure for an element.

```
OSStatus ISpElement_GetInfo (
                    const ISpElementReference inElement,
                    ISpElementInfoPtr outInfo);
```

`inElement`	A reference to the element whose element information structure you want to get.
`outInfo`	On return, a pointer to an element information structure. See `ISpElementInfo` (page 53) for more details.
function result	A result code. See “Result Codes” (page 93).

DISCUSSION

This function obtains information common to all elements (kind, label, and human-readable string). To get element kind–specific information use the `ISpElement_GetConfigurationInfo` function (page 21).

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElement_GetConfigurationInfo

Obtains configuration information for an element.

```
OSStatus ISpElement_GetConfigurationInfo (
                    const ISpElementReference inElement,
                    UInt32 buflen,
                    void *configInfo);
```

`inElement`	A reference to the element whose configuration information you want to get.
`buflen`	The length of the buffer for holding the information. The size of the configuration structures varies by element kind. If the buffer is not long enough to hold the data, the `ISpElement_GetConfigurationInfo` function copies as many bytes of data as fit and returns an error.
`configInfo`	A pointer to a buffer. On return, the buffer holds the requested configuration information.
function result	A result code. See "Result Codes" (page 93).

DISCUSSION

The `ISpElement_GetConfigurationInfo` function obtains information such as, for example, whether a directional pad has eight directions or four or whether to use a certain button first when matching game input requirements with controls. InputSprocket stores this type of information in configuration information structures that vary depending on the type of element—for example, the `ISpButtonConfigurationInfo` (page 54), `ISpDPadConfigurationInfo` (page 55), and `ISpAxisConfigurationInfo` (page 56) and `ISpDeltaConfigurationInfo` (page 56) structures.

VERSION NOTES

Introduced with InputSprocket 1.0.

Obtaining Data From Elements

You can use the functions in this section during game play to obtain data from the various elements.

- `ISpElement_GetSimpleState` (page 23) obtains the current state of an element whose data fits in an unsigned 32-bit integer.
- `ISpElement_GetComplexState` (page 24) obtains the current state of an element.
- `ISpElement_GetNextEvent` (page 24) obtains event data for a single element.

- `ISpElementList_GetNextEvent` (page 25) obtains the most recent event from a list of elements.
- `ISpUptime` (page 26) obtains the time elapsed since machine startup.
- `ISpTimeToMicroseconds` (page 27) converts from units of `AbsoluteTime` (as received in an InputSprocket event structure) to units of microseconds.
- `ISpElement_Flush` (page 27) flushes all the events on an element.
- `ISpElementList_Flush` (page 28) flushes all the events on an element list.

ISpElement_GetSimpleState

Obtains the current state of an element whose data fits in an unsigned 32-bit integer.

```
OSStatus ISpElement_GetSimpleState (
                    const ISpElementReference inElement,
                    Uint32 *state);
```

`inElement`	A reference to the element whose state you want to get.
`state`	The current state of the specified element. For a description of some values that this parameter can be set to, see “Button Element Data Information” (page 73), “Directional Pad Element Data Information” (page 73), and “Axis Element Data Information” (page 74).
function result	A result code. See “Result Codes” (page 93).

DISCUSSION

As most built-in elements return a 4-byte value or less, you should generally use this function in place of `ISpElement_GetComplexState` (page 24).

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElement_GetComplexState

Obtains the current state of an element.

```
OSStatus ISpElement_GetComplexState (
                     ISpElementReference inElement,
                     UInt32 buflen,
                     void *state);
```

`inElement`	A reference to the element whose state you want to get.
`buflen`	The length of the buffer allocated to hold the data. This must match the `dataSize` field of the element's element definition structure.
`state`	A pointer to a buffer to hold the data. On return, the state of the specified element.
function result	A result code. See "Result Codes" (page 93).

DISCUSSION

As most built-in element types return no larger than a 4-byte value, you should use the function `ISpElement_GetSimpleState` (page 23) instead.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElement_GetNextEvent

Obtains event data for a single element.

```
OSStatus ISpElement_GetNextEvent (
                     ISpElementReference inElement,
                     UInt32 bufSize,
                     ISpElementEventPtr event,
                     Boolean *wasEvent);
```

`inElement`	A reference to the element whose event data you want to get.
`bufSize`	The size of the buffer allocated to hold the event information.
`event`	A pointer to the buffer to hold the event data. On return, the buffer contains a structure of type `ISpElementEvent` (page 51). Note that this structure is of variable size depending on the event and the element involved.
`wasEvent`	On return, this value is `true` if there was an event; otherwise, it is `false`.
function result	A result code. See "Result Codes" (page 93).

DISCUSSION

Events for an element are stored in a queue in a first-in,first-out manner.

If there is not enough space to hold the entire event data structure, the event is removed from the event queue and `ISpElement_GetNextEvent` returns an error.

This function is the same as `ISpElementList_GetNextEvent` (page 25) except that it operates on a single element.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementList_GetNextEvent

Obtains the most recent event from a list of elements.

```
OSStatus ISpElementList_GetNextEvent (
                        ISpElementListReference inElementList,
                        UInt32 bufSize,
                        ISpElementEventPtr event,
                        Boolean *wasEvent);
```

`inElementList`	A reference to the element list to get the event from.
`bufSize`	The size of the buffer allocated to hold the event data.

`event`	A pointer to the buffer to hold the event data. On return, the buffer contains a structure of type `ISpElementEvent` (page 51). Note that this structure is of variable size depending on the event and the element involved.
`wasEvent`	On return, this value is `true` if there was an event; otherwise, it is `false`.
function result	A result code. See "Result Codes" (page 93).

DISCUSSION

Each element list has an event queue associated with it; events in the queue are stored in a first-in, first out manner.

If there is not enough space to hold the entire event data structure, the event is removed from the event queue and `ISpElement_GetNextEvent` returns an error.

This function is the same as `ISpElement_GetNextEvent` (page 24) except that it operates on an element list.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpUptime

Obtains the time elapsed since machine startup.

```
AbsoluteTime ISpUpTime (void);
```

function result	A value of type `AbsoluteTime` that indicates the time elapsed since the host computer was started up. Absolute time is a 64-bit monotonically increasing value. You should not make any assumptions about what units absolute time is based upon. On non-PCI machines, the `AbsoluteTime` value is interpolated from the elapsed time in ticks.

DISCUSSION

You can call this function from the task level, software interrupt level, or hardware interrupt level.

VERSION NOTES

Introduced with InputSprocket 1.1.

ISpTimeToMicroseconds

Converts from units of `AbsoluteTime` (as received in an InputSprocket event structure) to units of microseconds.

```
OSStatus ISpTimeToMicroseconds (
                    const AbsoluteTime *inTime,
                    UnsignedWide *outMicroseconds);
```

`inTime`	A pointer to the `AbsoluteTime` value you want to convert.
`outMicroseconds`	A pointer to a variable to hold the value in microseconds. On return, `outMicroseconds` points to the converted time value.
function result	A result code. See “Result Codes” (page 93).

VERSION NOTES

Introduced with InputSprocket 1.2.

ISpElement_Flush

Flushes the event queue associated with an element.

```
OSStatus ISpElement_Flush  (ISpElementReference inElement);
```

`inElement`	A reference to the element whose events you want to flush.

function result A result code. See “Result Codes” (page 93).

DISCUSSION

The `ISpElement_Flush` function guarantees to flush only those events that made it to the InputSprocket layer before the call. It will not flush any events that made it to the InputSprocket layer after the call returns. The outcome for events that occur during the call is undefined.

The `ISpElement_Flush` function is the same as `ISpElementList_Flush` except that it operates on a single element.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementList_Flush

Flushes the event queue associated with an element list.

```
OSStatus ISpElementList_Flush  (ISpElementListReference inElementList);
```

`inElementList`
A reference to the list of elements whose events you want to flush.

function result A result code. See “Result Codes” (page 93).

DISCUSSION

The `ElementList Flush` function guarantees to flush only those events that made it to the InputSprocket layer before the call. It will not flush any events that made it to the InputSprocket layer after the call returns. The outcome for events that occur during the call is undefined.

The `ISpElementList_Flush` function is the same as `ISpElement_Flush` except that it operates on an element list.

VERSION NOTES

Introduced with InputSprocket 1.0.

Manipulating Devices

This section describes functions used to activate, deactivate, and get information about input devices connected to the host computer.

- `ISpDevices_Extract` (page 29) extracts and counts devices listed on the systemwide list of devices.
- `ISpDevices_ExtractByClass` (page 30) extracts and counts devices of a certain class listed on the systemwide list of devices.
- `ISpDevices_ExtractByIdentifier` (page 32) extracts and counts devices of a certain type that are listed on the systemwide list of devices.
- `ISpDevices_Activate` (page 33) activates the specified devices.
- `ISpDevices_Deactivate` (page 33) deactivates the specified devices.
- `ISpDevices_ActivateClass` (page 34) activates the specified class of devices.
- `ISpDevices_DeactivateClass` (page 35) deactivates the specified class of devices.
- `ISpDevice_IsActive` (page 36) finds out if a device is active.
- `ISpDevice_GetDefinition` (page 36) obtains a device definition structure for a specified device.
- `ISpDevice_GetElementList` (page 37) obtains an element list for a specified device.

ISpDevices_Extract

Extracts and counts devices listed on the systemwide list of devices.

```
OSStatus ISpDevices_Extract (
                    UInt32 inBufferCount,
                    UInt32 *outCount,
                    ISpDeviceReference *buffer);
```

`inBufferCount`	The number of device references in the array pointed to by the `buffer` parameter.
`outCount`	A pointer to an integer variable. On return, the variable specifies the number of device references on the systemwide list.
`buffer`	A pointer to an array of device references. On return, `buffer` points to the array of extracted device references.
function result	A result code. See "Result Codes" (page 93).

DISCUSSION

Extracting devices may be useful if you want to find and activate input devices—usually the keyboard and mouse—prior to autoconfiguration. The `ISpDevices_Extract` function copies device references from the systemwide list of devices into the array specified by `buffer`. If there are more devices in the list than there is space in your array, it copies only as many device references as fit. The function sets the `outCount` parameter to the total number of devices in the system wide list of devices.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpDevices_ExtractByClass

Extracts and counts devices of a certain class listed on the systemwide list of devices.

```
OSStatus ISpDevices_ExtractByClass (
                        ISpDeviceClass theClass,
                        UInt32 inBufferCount,
                        UInt32 *outCount,
                        ISpDeviceReference *buffer);
```

`theClass`	The category of device to count and extract. See "Built-in Device Categories" (page 59) for built-in values for this parameter.

`inBufferCount`	The number of device references in the array pointed to by the `buffer` parameter.
`outCount`	A pointer to an integer variable. On return, the variable specifies the number of device references on the systemwide list.
`buffer`	A pointer to an array of device references. On return, `buffer` points to the array of extracted device references.
function result	A result code. See "Result Codes" (page 93).

DISCUSSION

If you want to extract classes of devices so you can activate or deactivate them, you should use the functions `ISpDevices_ActivateClass` (page 34) or `ISpDevices_DeactivateClass` (page 35) instead.

Extracting devices may be useful if you want to find and activate input devices—usually the keyboard and mouse—prior to autoconfiguration. The `ISpDevices_ExtractByClass function` copies into that array specified by `buffer` the device references of the class specified by the `theClass` parameter found on the systemwide list of devices. If there are more devices of that class in the list than there is space in your array, it copies only as many device references as fit. The function sets the `outCount` parameter to the total number of devices of the specified category in the systemwide list of devices.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpDevices_ExtractByIdentifier

Extracts and counts devices of a certain type that are listed on the systemwide list of devices.

```
OSStatus ISpDevices_ExtractByIdentifier (
                        ISpDeviceIdentifier theIdentifier,
                        UInt32 inBufferCount,
                        UInt32 *outCount,
                        ISpDeviceReference *buffer);
```

`theIdentifier`	The type of device to count and extract.
`inBufferCount`	The number of device references in the array pointed to by the `buffer` parameter.
`outCount`	The number of devices of the specified type on the systemwide list.
`buffer`	A pointer to an array of device references.
function result	A result code. See "Result Codes" (page 93).

DISCUSSION

Extracting devices may be useful if you want to find and activate input devices—usually the keyboard and mouse—prior to autoconfiguration. The `ISpDevices_ExtractByIdentifier function` copies into the array specified by `buffer` the device references of the type specified by the `theIdentifier` parameter found on the systemwide list of devices. If there are more devices of that type in the list than there is space in the array, it copies only as many device references as fit. The function sets the `outCount` parameter to the total number of devices of the specified type in the systemwide list of devices.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpDevices_Activate

Activates the specified devices.

```
OSStatus ISpDevices_Activate (
                    UInt32 inDeviceCount,
                    ISpDeviceReference *inDevicesToActivate);
```

`inDeviceCount`
: The number of references in the array pointed to by the `inDevicesToActivate` parameter.

`inDevicesToActivate`
: A pointer to an array of device references that correspond to the devices you want to activate.

function result A result code. See “Result Codes” (page 93).

DISCUSSION

When a device is activated, InputSprocket receives events from it.

The following categories of devices are inactive by default: `kIspDeviceClass_SpeechRecognition`, `kISpDeviceClass_Mouse`, and `kISpDeviceClass_Keyboard`.

If you want to activate classes of devices, you should use the function `ISpDevices_ActivateClass` (page 34) instead.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpDevices_Deactivate

Deactivates the specified devices.

```
OSStatus ISpDevices_Deactivate (
                    UInt32 inDeviceCount,
                    ISpDeviceReference *inDevicesToDeactivate);
```

`inDeviceCount`
: The number of references in the array pointed to by the `inDevicesToActivate` parameter.

`inDevicesToDeactivate`
: A pointer to an array of device references that correspond to the devices you want to deactivate.

function result A result code. See "Result Codes" (page 93).

DISCUSSION

When a device is deactivated, InputSprocket no longer receives events from it.

For example, you might want to get input from the keyboard or mouse as text style data. All devices in the system start out activated, except the mouse, the keyboard, and speech recognition.

If you want to deactivate classes of devices, you should use the function `ISpDevices_DeactivateClass` (page 35) instead.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpDevices_ActivateClass

Activates the specified class of devices.

```
OSStatus ISpDevices_ActivateClass (ISpDeviceClass inClass);
```

`inClass` A value of type `ISpDeviceClass` that specifies the class of device you want to activate.

function result A result code. See "Result Codes" (page 93).

DISCUSSION

In most cases it is easier to activate classes of devices rather than extracting them and activating them individually by calling `ISpDevices_Activate` (page 33).

VERSION NOTES

Introduced with InputSprocket 1.1.

ISpDevices_DeactivateClass

Deactivates the specified class of devices.

```
OSStatus ISpDevices_DeactivateClass (ISpDeviceClass inClass);
```

`inClass`	A value of type `ISpDeviceClass` that specifies the class of device you want to deactivate.
function result	A result code. See “Result Codes” (page 93).

DISCUSSION

In most cases it is easier to deactivate classes of devices rather than extracting them and activating them individually by calling `ISpDevices_Deactivate` (page 33).

VERSION NOTES

Introduced with InputSprocket 1.1.

ISpDevice_IsActive

Finds out if a device is active.

```
OSStatus ISpDevice_IsActive (
                    ISpDeviceReference inDevice,
                    Boolean *outIsActive);
```

`inDevice`	A reference to a device.
`outIsActive`	This value is `true` if the device is active, `false` if it is inactive.
function result	A result code. See “Result Codes” (page 93).

DISCUSSION

Note that all devices are active by default except for speech recognition, mice, and the keyboard.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpDevice_GetDefinition

Obtains a device definition structure for a specified device.

```
OSStatus ISpDevice_GetDefinition (
                    const ISpDeviceReference inDevice,
                    UInt32 buflen,
                    ISpDeviceDefinition *outStruct);
```

`inDevice`	The device whose device definition structure you want to get.
`buflen`	The length of buffer to hold the device definition.
`outStruct`	A pointer to a device definition structure. See `ISpDeviceDefinition` (page 52) for more information.
function result	A result code. See “Result Codes” (page 93).

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpDevice_GetElementList

Obtains an element list for a specified device.

```
OSStatus ISpDevice_GetElementList (
                     const ISpDeviceReference inDevice,
                     ISpElementListReference *outElementList);
```

`inDevice`	A reference to the device whose element list you want to get.
`outElementList`	On return, a reference to the element list. Note that you should not add or remove elements from this list, and you must not dispose of the list.
function result	A result code. See "Result Codes" (page 93).

VERSION NOTES

Introduced with InputSprocket 1.0.

Managing Element Lists

Use the functions in this section to create element lists and to add, remove, and extract elements from those lists.

- `ISpElementList_New` (page 38) creates a new element list.
- `ISpElementList_Dispose` (page 39) disposes of a specified element list.
- `ISpGetGlobalElementList` (page 39) obtains a global element list, which is is a list of all the elements in the system.
- `ISpElementList_AddElements` (page 40) adds more elements to an element list.
- `ISpElementList_RemoveElements` (page 41) removes elements from an element list.

- `ISpElementList_Extract` (page 42) extracts and counts elements in an element list.
- `ISpElementList_ExtractByKind` (page 43) extracts and counts elements of a specified kind in an element list.
- `ISpElementList_ExtractByLabel` (page 44) extracts and counts elements with a specific label in an element list.

ISpElementList_New

Creates a new element list.

```
OSStatus ISpElementList_New (
                    UInt32 inCount,
                    ISpElementReference *inElements,
                    ISpElementListReference *outElementList
                    UInt32 flags);
```

`inCount`	The number of element references in the list pointed to by the `inElements` parameter. If you pass in 0 in the `inCount` parameter, `ISpElementList_New` creates an empty list.
`inElements`	A pointer to an array of element references corresponding to the elements to put in the list.
`outElementList`	A pointer to a reference to the new element list. If `ISpElementList_New` fails to create a new list because it was out of memory, the function sets this parameter to 0.
`flags`	If you want the list allocated in temporary memory, pass in `kISpElementListFlag_UseTempMem` in the `flags` parameter. See “Element List Flag” (page 75).
function result	A result code. See “Result Codes” (page 93).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementList_Dispose

Disposes of a specified element list.

```
OSStatus ISpElementList_Dispose  (ISpElementListReference inElementList);
```

`inElementList`	A reference to the element list you want to dispose.
function result	A result code. See "Result Codes" (page 93).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpGetGlobalElementList

Obtains a global element list, which is a list of all the elements in the system.

```
OSStatus ISpGetGlobalElementList (
                    ISpElementListReference *outElementList);
```

`outElementList`	A reference to the global element list.
function result	A result code. See "Result Codes" (page 93).

DISCUSSION

You cannot modify this element list, and you must not attempt to dispose it.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementList_AddElements

Adds more elements to an element list.

```
OSStatus ISpElementList_AddElements (
                    ISpElementListReference inElementList,
                    UInt32 refCon,
                    UInit32 count,
                    ISpElementReference *newElements);
```

`inElementList`	A reference to the element list you want to add elements to.
`refCon`	On output, a reference constant used to locate the new elements in the element list.
`count`	The number of elements to be added.
`newElements`	A pointer to the array of element references you want to add.
function result	A result code. See “Result Codes” (page 93).

DISCUSSION

The function sets the `refCon` parameter to a reference constant to be used in identifying the new elements in the list.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementList_RemoveElements

Removes elements from an element list.

```
OSStatus ISpElementList_RemoveElements (
                    ISpElementListReference inElementList,
                    UInit32 count,
                    ISpElementReference *oldElement);
```

`inElementList`	A reference to the element list containing the elements you want to remove.
`count`	The number of elements to remove from the list.
`oldElement`	A pointer to a block of element references that indicates the elements you want to remove.
function result	A result code. See “Result Codes” (page 93).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementList_Extract

Extracts and counts elements in an element list.

```
OSStatus ISpElementList_Extract (
                    ISpElementListReference inElementList,
                    UInt32 inBufferCount,
                    UInt32 *outCount,
                    ISpElementReference *buffer);
```

`inElementList`	A reference to the element list to extract from.
`inBufferCount`	The number of element references in the array pointed to by the `buffer` parameter.
`outCount`	On output, the number of element references on the element list.
`buffer`	On output, a pointer to an array of element references indicating the extracted elements.
function result	A result code. See "Result Codes" (page 93).

DISCUSSION

The `ISpElementList_Extract` function takes, in the `buffer` parameter, a pointer to an array of element references and, in the `inBufferCount` parameter, the number of element references in the array. The `ISpElementList_Extract` function copies element references from the list specified in the `inElementList` parameter into that array. If there are more elements in the list than there is space in your array, it copies only as many element references as fit. The function sets the `outCount` parameter to the total number of elements in the element list.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementList_ExtractByKind

Extracts and counts elements of a specified kind in an element list.

```
OSStatus ISpElementList_ExtractByKind (
                        ISpElementListReference inElementList,
                        ISpElementKind theKind,
                        UInt32 inBufferCount,
                        UInt32 *outCount,
                        ISpElementReference *buffer);
```

`inElementList`	A reference to the element list containing the elements you want to extract.
`theKind`	The kind of elements to extract.
`inBufferCount`	On input, the number of element references in the array pointed to by the `buffer` parameter. On output, `inBufferCount` holds the number of element references that match the specified kind.
`outCount`	On output , the number of element references of the specified kind in the element list that were actually copied to the buffer.
`buffer`	On output, a pointer to an array of element references indicating the elements that match the specified kind.
function result	A result code. See “Result Codes” (page 93).

DISCUSSION

The `ISpElementList_ExtractByKind` function takes, in the `buffer` parameter, a pointer to an array of element references and, in the `inBufferCount` parameter, the number of element references in the array. The `ISpElementList_ExtractByKind` function copies element references of the kind specified by the `theKind` parameter from the list specified in the `inElementList` parameter into the array pointed to by `buffer`. If there are more elements of the specified kind in the list than there is space in the array, it copies only as many element references as fit. The function sets the `outCount` parameter to the total number of elements of the specified kind in the element list.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementList_ExtractByLabel

Extracts and counts elements with a specific label in an element list.

```
OSStatus ISpElementList_ExtractByLabel (
                    ISpElementListReference inElementList,
                    ISpElementLabel theLabel,
                    UInt32 inBufferCount,
                    UInt32 *outCount,
                    ISpElementReference *buffer);
```

`inElementList`	A reference to the element list containing the elements you want to extract.
`theLabel`	The label of elements to extract.
`inBufferCount`	On input, the number of element references in the array pointed to by the `buffer` parameter. On output, `inBufferCount` holds the number of element references that match the specified label.
`outCount`	On output, the number of element references with the specified label in the element list that were actually copied to the buffer*.
`buffer`	On output, a pointer to an array of element references indicating the elements that match the specified label.
function result	A result code. See "Result Codes" (page 93).

DISCUSSION

The `ISpElementList_ExtractByLabel` function takes, in the `buffer` parameter, a pointer to an array of element references and, in the `inBufferCount` parameter, the number of element references in the array. The function copies element references with the label specified by the `theLabel` parameter from the list specified in the `inElementList` parameter into that array. If there are more elements of the specified label in the list than there is space in the array, it copies only as many element references as fit. The function sets the `outCount` parameter tothe total number of elements with the specified label in the element list.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with InputSprocket 1.0.

Application-Defined Function

This section describes the application-defined function `MyEventProc` (page 45), which you use to handle events during calls to the function `ISpConfigure` (page 13).

MyEventProc

Handles events during calls to the function `ISpConfigure` (page 13).

```
Boolean MyEventProc (EventRecord *inEvent);
```

`inEvent`	A pointer to an event record that describes the event that occurred.
function result	If your function handled the event, it should return True; otherwise it should return False.

DISCUSSION

When calling the function `ISpConfigure` (page 13), you can designate an application-defined function to handle events that may occur while the configuration window is active.

VERSION NOTES

Introduced with InputSprocket 1.0.

Data Types

This section describes the data structures provided by InputSprocket.

- `ISpDeviceReference` (page 47)
- `ISpElementReference` (page 47)
- `ISpElementListReference` (page 47)
- `ISpDeviceClass` (page 48)
- `ISpDeviceIdentifier` (page 48)
- `ISpNeed` (page 48)
- `ISpElementLabel` (page 50)
- `ISpElementKind` (page 50)
- `ISpEventProcPtr` (page 51)
- `ISpElementEvent` (page 51)
- `ISpDeviceDefinition` (page 52)
- `ISpElementInfo` (page 53)
- `ISpButtonConfigurationInfo` (page 54)
- `ISpDPadConfigurationInfo` (page 55)
- `ISpAxisConfigurationInfo` (page 56)
- `ISpDeltaConfigurationInfo` (page 56)
- `ISpMovementConfigurationInfo` (page 57)

- `ISpMovementData` (page 57)
- `ISpApplicationResourceStruct` (page 58)

ISpDeviceReference

InputSprocket handles devices by passing a device reference, which is defined by the `ISpDeviceReference` type:

```
typedef struct OpaqueISpDeviceReference*  ISpDeviceReference;
```

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementReference

InputSprocket handles virtual elements by passing an element reference, which is defined by the `ISpElementReference` type:

```
typedef struct OpaqueISpElementReference*  ISpElementReference;
```

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementListReference

InputSprocket handles element lists by passing an element list reference, which is defined by the `ISpElementReferenceList` type:

```
typedef struct OpaqueISpElementListReference*  ISpElementListReference;
```

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpDeviceClass

InputSprocket handles classes of input devices by passing values of type `ISpDeviceClass`:

```
typedef OSType ISpDeviceClass;
```

Examples of device classes would be keyboards, mice, or joysticks.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpDeviceIdentifier

InputSprocket handles physical input devices by passing values of type `ISpDeviceIdentifier`:

```
typedef OSType ISpDeviceIdentifier;
```

Values of this type represent a specific device, such as a one-button mouse or a 105-key extended keyboard.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpNeed

During initialization the game fills out a need structure for each input requirement. The need structure describes the type of data that will satisfy the input requirement and also gives information you can use in a user interface during configuration. The need structure is defined by the `ISpNeed` data type.

```
typedef struct ISpNeed {
    Str63               name;
    short               iconSuiteResourceId;
    UInt8               playerNum
    UInt8               group
    ISpElementKind      theKind;
    ISpElementLabel     theLabel;
    UInt32              flags;
    UInt32              reserved1;
    UInt32              reserved2;
    UInt32              reserved3;
} ISpNeed;
```

Field Descriptions

`name`	A human-readable string that can be used during configuration to describe the input requirement to the user.
`iconSuiteResourceId`	A resource ID of an icon suite residing in the games resource fork. This is the same resource ID you would pass to the `GetIconSuite` function if you wanted to load the icon yourself. The icon is a picture representing the input requirement.
`playerNum`	An integer indicating the player to whom this element belongs. A value of 0 indicates that the element belongs to no player (for example, a button to end the game affects all players, so it would not be associated with any single player). Otherwise, 1 indicates player 1, 2 indicates player 2, and so on.
`group`	An integer indicating to which group this element belongs. A group is any arbitrary set which may be useful to define for game purposes (for example, you could define teams of players). A common convention is to group elements that control common needs together (such as left/right buttons and an axis control, both of which control left-to-right movement).
`theKind`	The kind of element that will produce the data the game requires. This can be a standard element kind—for example, `kISpElementKind_Button`—or an extended virtual kind. The only extended kind that exists currently is the movement kind, which consists of an x-y axis pair and a

	direction, where both are based on the same data and the game uses whichever is appropriate.
`theLabel`	A standard element label representing what you are going to use the data for.
`flags`	You should OR in `kISpNeedFlag_NoMultiConfig` if you would prefer that only one device bind to the need during autoconfiguration. Otherwise, several devices may agree to meet the same input requirement.
`reserved1`	Reserved. Always set this field to 0.
`reserved2`	Reserved. Always set this field to 0.
`reserved3`	Reserved. Always set this field to 0.

VERSION NOTES

This version of the need structure introduced with InputSprocket 1.2.

ISpElementLabel

InputSprocket handles element labels by passing values of type `ISpElementLabel`:

```
typedef OSType ISpElementLabel;
```

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementKind

InputSprocket handles element kinds by passing values of type `ISpElementKind`:

```
typedef OSType ISpElementKind;
```

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpEventProcPtr

When calling the function `ISpConfigure` (page 13), you must designate an application-defined function to handle events. Such a function has the following type definition:

```
typedef Boolean (*ISpEventProcPtr)(EventRecord *inEvent);
```

See `MyEventProc` (page 45) for more information about implementing this function.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementEvent

The element event structure is a variable length structure that passes element event data. An element event data structure is defined by the `ISpElementEvent` data type.

```
typedef struct {
    AbsoluteTime        when;
    ISpElementReference element;
    UInt32              refCon;
    Uint32              data;
} ISpElementEvent, *ISpElementEventPtr;
```

Field Descriptions

`when`	The time the event happened. If the application is running on a PCI machine, the time will be in units of `AbsoluteTime` (as generated by the `ISpUptime` (page 26) call). Otherwise, the `when.hi` field will be 0 and the `when.lo` field will be the time generated by `TickCount`.
`element`	A reference to the element that generated the event.
`refCon`	The reference constant assigned to this element on the element list that contains it (see the

	ISpElementList_AddElements function on (page 40). If you got this event from the ISpElement_GetNextEvent function or from a global list created by the system, this field is 0.
data	The data for the event. It is a variable length field that is often, but not always, a 32-bit unsigned integer. Valid values include those given in the description of the "Button Element Data Information" (page 73), "Directional Pad Element Data Information" (page 73), "Axis Element Data Information" (page 74), and in the ISpMovementData (page 57) data structure. You need to examine the element kind to determine what size and type of data is reported. The type of data varies for different elements. For example, a new type of input data may refer to RGB color, so it requires 6 bytes of data instead of 4 bytes. It would be typed by a new element kind—for example, 'rgbc'. If you wanted to get RGB element kind data, you would have to allocate a larger structure when you were getting events via the event mechanism than you would allocate for 4-byte data.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpDeviceDefinition

A device definition structure provides all the information about the input device available within the system. A device definition structure is defined by the ISpDeviceDefinition data type.

```
typedef struct ISpDeviceDefinition {
    Str63                   deviceName;
    ISpDeviceClass          theDeviceClass;
    ISpDeviceIdentifier     theDeviceIdentifier;
    UInt32                  permanentID;
    UInt32                  flags;
    UInt32                  permanentIDExtend;
    UInt32                  reserved2;
    UInt32                  reserved3;
} ISpDeviceDefinition;
```

Field Descriptions

`deviceName`	A human-readable string with the name of the device.
`theDeviceClass`	The general category of device—for example, joystick, keyboard, mouse, and so on. There are built-in values for the `theDeviceClass` parameter, and device driver developers may create new ones. For a list of built-in values see “Built-in Device Categories” (page 59).
`theDeviceIdentifier`	The registered four-character sequence that identifies the type of device.
`permanentID`	An ID that identifies the device even after rebooting. If the device driver cannot distinguish this device from other devices with the same value for `theDeviceIdentifier`, this field should be 0.
`flags`	Status flags.
`permanentIDExtend`	An extension of the `permanentID` field, so the ID can be up to 64 bits long.
`reserved2`	Reserved. Always set this field to 0.
`reserved3`	Reserved. Always set this field to 0.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpElementInfo

The element information structure provides basic information about an element that is common to all elements, regardless of kind. An element information structure is defined by the `ISpElementInfo` data type.

```
typedef struct ISpElementInfo {
    ISpElementLabel     theLabel;
    ISpElementKind      theKind;
    Str63               theString;
    UInt32              reserved1;
    UInt32              reserved2;
} ISpElementInfo, *ISpElementInfoPtr;
```

Field Descriptions

`theLabel`	The label of the element. See “Element Label Constants” (page 62) for built-in values.
`theKind`	The kind of data produced by the element. See “Built-in Element Kinds” (page 61) for built-in values.
`theString`	The localized human-readable identifier for use in the user interface.
`reserved1`	Reserved. Always set this field to 0.
`reserved2`	Reserved. Always set this field to 0.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpButtonConfigurationInfo

The button configuration information structure provides information used during configuration and in interpreting button element data. For each element of kind `kISpElementKind_Button`, the device driver fills out a button configuration information structure, which is stored by InputSprocket. A button configuration information structure is defined by the `ISpButtonConfigurationInfo` data type.

```
typedef struct {
    UInt32              id;
} ISpButtonConfigurationInfo;
```

Field Descriptions

`id`	Use this ID to indicate which button element to assign first during configuration if the device has more than one button element. A lower value indicates priority. For example, an easy-to-reach button should have a low value (such as 1), so that it will be assigned first, while one that is less accessible should be given a high value (for example, 6). A value of 0 indicates no assignment priority.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpDPadConfigurationInfo

The directional pad configuration information structure provides information used during configuration and in interpreting directional pad element data. For each element of kind `kISpElementKind_DPad`, the device driver fills out a directional pad configuration information structure, which is stored by InputSprocket. A directional pad configuration information structure is defined by the `ISpDPadConfigurationInfo` data type.

```
typedef struct {
    UInt32              id;
    Boolean             fourWayPad;
} ISpDPadConfigurationInfo;
```

Field Descriptions

`id`	Use this ID to indicate which directional pad to assign first during configuration if the device has more than one directional pad element. A lower value indicates priority. For example, an easy-to-reach pad should have a low value (such as 1), so that it will be assigned first, while one that is less accessible should be given a high value (for example, 6). A value of 0 indicates no assignment priority.
`fourWayPad`	This value is `true` if the pad can produce only four directions plus idle.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpAxisConfigurationInfo

The axis configuration information structure provides information used during configuration and in interpreting axis element data. For each element of kind `kISpElementKind_Axis`, the device driver fills out an axis configuration information structure, which is stored by InputSprocket. An axis configuration information structure is defined by the `ISpAxisConfigurationInfo` data type.

```
typedef struct {
    Boolean             SymmetricAxis;
} ISpAxisConfigurationInfo;
```

Field Descriptions

`SymmetricAxis`	This value is `true` if the axis has a meaningful center—for example, the axis of a joystick. In this case, 0 or idle falls at the `kISpAxisMiddle` position. This value is `false` if the axis has no meaningful center—for example, the axis of a brake or gas pedal.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpDeltaConfigurationInfo

The delta configuration information structure currently provides no information and is included for completeness only. Additional information may appear in future versions of InputSprocket.

```
typedef struct ISpDeltaConfigurationInfo {
    UInt32 reserved1;
    UInt32 reserved2;
    } ISpDeltaConfigurationInfo;
```

Field Descriptions

`reserved1`	Reserved.
`reserved2`	Reserved.

VERSION NOTES

Introduced with InputSprocket 1.2.

ISpMovementConfigurationInfo

The movement configuration information structure currently provides no information and is included for completeness only. Additional information may appear in future versions of InputSprocket.

```
typedef struct ISpMovementConfigurationInfo {
    UInt32 reserved1;
    UInt32 reserved2;
    }; ISpMovementConfigurationInfo;
```

Field Descriptions

`reserved1`	Reserved.
`reserved2`	Reserved.

VERSION NOTES

Introduced with InputSprocket 1.1.

ISpMovementData

Note that in most cases you should avoid using movement elements and use axis elements instead.

The movement data structure provides data from elements of kind `kISpElementKind_Movement`. This element kind produces data that is given both as x-y axis data and directional pad data, allowing the game to use whichever is suitable. The movement data structure is defined by the ISpMovementData data type.

```
typedef struct ISpMovementData {
    UInt32              xAxis;
    UInt32              yAxis;
    UInt32              direction;
};
```

Field Descriptions

xAxis	Movement data given in terms of the x-axis of an x-y axis pair.
yAxis	Movement data given in terms of the y-axis of an x-y axis pair.
direction	Movement data given as a direction.

VERSION NOTES

Introduced with InputSprocket 1.0.

ISpApplicationResourceStruct

You use this structure to identify what features of InputSprocket the application is using. This structure is typically used only by third-party driver software that may want to know how the application uses InputSprocket.

```
typedef struct ISpApplicationResourceStruct {
    UInt32                      flags;
    UInt32                      reserved1;
    UInt32                      reserved2;
    UInt32                      reserved3;
}; ISpApplicationResourceStruct;
```

Field Descriptions

flags	A bit field containing flags that describe the application. See “Application Resource Constants” (page 77) for a list of possible values.

`reserved1`	Reserved. Set to 0.
`reserved2`	Reserved. Set to 0.
`reserved3`	Reserved. Set to 0.

VERSION NOTES

Introduced with InputSprocket 1.2.

Constants

This section describes the constants provided by InputSprocket.

Built-in Device Categories

These constants identify the general category of devices the input device belongs to. Use the constants in the `theDeviceClass` field of the `ISpDeviceDefinition` data structure.

```
enum {
    kISpDeviceClass_SpeechRecognition       = FOUR_CHAR_CODE('talk'),
    kISpDeviceClass_Mouse                   = FOUR_CHAR_CODE('mous'),
    kISpDeviceClass_Keyboard                = FOUR_CHAR_CODE('keyd'),
    kISpDeviceClass_Joystick                = FOUR_CHAR_CODE('joys'),
    kISpDeviceClass_Gamepad                 = FOUR_CHAR_CODE('gmpd'),
    kISpDeviceClass_Wheel                   = FOUR_CHAR_CODE('whel'),
    kISpDeviceClass_Pedals                  = FOUR_CHAR_CODE('pedl'),
    kISpDeviceClass_Levers                  = FOUR_CHAR_CODE('levr'),
    kISpDeviceClass_Tickle                  = FOUR_CHAR_CODE('tckl'),
    kISpDeviceClass_Unknown                 = FOUR_CHAR_CODE('????')
};
```

Constant Descriptions

`kIspDeviceClass_SpeechRecognition`
: The device is primarily a speech-recognition device.

`kISpDeviceClass_Mouse`
: The device is a one-button mouse.

`kISpDeviceClass_Keyboard`
: The device is a keyboard.

`kISpDeviceClass_Joystick`
: The device is a joystick.

`kISpDeviceClass_Gamepad`
: The device is a gamepad.

`kISpDeviceClass_Wheel`
: The device is primarily a wheel.

`kISpDeviceClass_Pedals`
: The device is primarily a pedal.

`kISpDeviceClass_Levers`
: The device is primarily a lever—for example, a device built around a thrust lever.

`kISpDeviceClass_Tickle`
: The device requires calls to `ISpTickle` (page 16) in order to operate (for example, speech recognition).

`kISpDeviceClass_Unknown`
: The device is of an unknown class.

VERSION NOTES

Introduced with InputSprocket 1.0.

Built-in Element Kinds

Use these constants to specify the kind of data an element produces and to identify virtual elements. Element kind constants are used in the ISpElementInfo structure and by the ISpElementList_ExtractByKind function.

```
enum {
    kISpElementKind_Button      = FOUR_CHAR_CODE('butn'),
    kISpElementKind_DPad        = FOUR_CHAR_CODE('dpad'),
    kISpElementKind_Axis        = FOUR_CHAR_CODE('axis'),
    kISpElementKind_Delta       = FOUR_CHAR_CODE('dlta'),
    kISpElementKind_Movement    = FOUR_CHAR_CODE('move'),
    kISpElementKind_Virtual     = FOUR_CHAR_CODE('virt')
};
```

Constant Descriptions

kISpElementKind_Button
Button data.

kISpElementKind_DPad
Directional pad data.

kISpElementKind_Axis
Axis data, either with or without a meaningful center position (as determined by the ISpAxisConfigurationInfo data structure).

kISpElementKind_Delta
Delta data, which indicates the distance moved relative to the previous position.

kISpElementKind_Movement
Movement data that is given both as x-y axis data and directional pad data, allowing the game to use whichever is suitable. Note that in general you should use axis data instead.

kISpElementKind_Virtual
A virtual element created by the ISpElement_NewVirtual function. Those created by the

ISpElement_NewVirtualFromNeeds function have the element kind specified by the need structure they correspond to.

VERSION NOTES

Introduced with InputSprocket 1.0.

Element Label Constants

You use element label constants to indicate the intended usage of an element in a need structure (that is a structure of type ISpElementInfo). InputSprockets uses these constants to help autoconfigure devices and to determine which labels appear as valid choices for certain controls.

InputSprocket provides element label constants for each of the built-in element kinds:

- "Generic Label Constant" (page 62)
- "Axis Label Constants" (page 63)
- "Delta Label Constants" (page 65)
- "Directional Pad Element Constants" (page 66)
- "Button Element Constants" (page 67)

Generic Label Constant

If an element has no preferred usage, you can assign the following constant in its need structure:

```
enum {
    kISpElementLabel_None               = FOUR_CHAR_CODE('none')
};
```

Constant Description

kISpElementLabel_None
A generic label; this control may be used for anything.

VERSION NOTES

Introduced with InputSprocket 1.0.2.

Axis Label Constants

You can use the following constants to indicate preferred usage for axis elements:

```
enum {
    kISpElementLabel_Axis_XAxis         = FOUR_CHAR_CODE('xaxi'),
    kISpElementLabel_Axis_YAxis         = FOUR_CHAR_CODE('yaxi'),
    kISpElementLabel_Axis_ZAxis         = FOUR_CHAR_CODE('zaxi'),
    kISpElementLabel_Axis_Rx            = FOUR_CHAR_CODE('rxax'),
    kISpElementLabel_Axis_Ry            = FOUR_CHAR_CODE('ryax'),
    kISpElementLabel_Axis_Rz            = FOUR_CHAR_CODE('rzax'),
    kISpElementLabel_Axis_Roll          = kISpElementLabel_Axis_Rz,
    kISpElementLabel_Axis_Pitch         = kISpElementLabel_Axis_Rx,
    kISpElementLabel_Axis_Yaw           = kISpElementLabel_Axis_Ry,
    kISpElementLabel_Axis_RollTrim      = FOUR_CHAR_CODE('rxtm'),
    kISpElementLabel_Axis_PitchTrim     = FOUR_CHAR_CODE('trim'),
    kISpElementLabel_Axis_YawTrim       = FOUR_CHAR_CODE('rytm'),
    kISpElementLabel_Axis_Gas           = FOUR_CHAR_CODE('gasp'),
    kISpElementLabel_Axis_Brake         = FOUR_CHAR_CODE('brak'),
    kISpElementLabel_Axis_Clutch        = FOUR_CHAR_CODE('cltc'),
    kISpElementLabel_Axis_Throttle      = FOUR_CHAR_CODE('thrt'),
    kISpElementLabel_Axis_Trim          = kISpElementLabel_Axis_PitchTrim,
    kISpElementLabel_Axis_Rudder        = FOUR_CHAR_CODE('rudd'),
    kISpElementLabel_Axis_ToeBrake      = FOUR_CHAR_CODE('toeb')
};
```

Constant Descriptions

`kISpElementLabel_Axis_XAxis`
The element should be an x-axis control.

`kISpElementLabel_Axis_YAxis`
The element should be a y-axis control.

`kISpElementLabel_Axis_ZAxis`
The element should be a z-axis control.

`kISpElementLabel_Axis_Rx`
The element should control rotation about the x axis.

`kISpElementLabel_Axis_Ry`	The element should control rotation about the y axis.
`kISpElementLabel_Axis_Rz`	The element should control rotation about the z axis.
`kISpElementLabel_Axis_Roll`	The element should control the roll of a craft.
`kISpElementLabel_Axis_Pitch`	The element should control the pitch of a craft.
`kISpElementLabel_Axis_Yaw`	The element should control the yaw of a craft.
`kISpElementLabel_Axis_RollTrim`	The element should trim the roll of a craft.
`kISpElementLabel_Axis_PitchTrim`	The element should trim the pitch of a craft.
`kISpElementLabel_Axis_YawTrim`	The element should trim the yaw of a craft.
`kISpElementLabel_Axis_Gas`	The element should control a gas pedal.
`kISpElementLabel_Axis_Brake`	The element should control a brake.
`kISpElementLabel_Axis_Clutch`	The element should control a clutch.
`kISpElementLabel_Axis_Throttle`	The element should control a throttle.
`kISpElementLabel_Axis_Trim`	The element should be a trim control.
`kISpElementLabel_Axis_Rudder`	The element should control a rudder.
`kISpElementLabel_Axis_ToeBrake`	The element should control an aircraft's toebrake.

VERSION NOTES

These element-based labels introduced with InputSprocket 1.0.2.

Delta Label Constants

You can use the following constants to indicate preferred usage for delta elements:

```
enum {
    kISpElementLabel_Delta_X          = FOUR_CHAR_CODE('xdlt'),
    kISpElementLabel_Delta_Y          = FOUR_CHAR_CODE('ydlt'),
    kISpElementLabel_Delta_Z          = FOUR_CHAR_CODE('zdlt'),
    kISpElementLabel_Delta_Rx         = FOUR_CHAR_CODE('rxdl'),
    kISpElementLabel_Delta_Ry         = FOUR_CHAR_CODE('rydl'),
    kISpElementLabel_Delta_Rz         = FOUR_CHAR_CODE('rzdl'),
    kISpElementLabel_Delta_Roll       = kISpElementLabel_Delta_Rz,
    kISpElementLabel_Delta_Pitch      = kISpElementLabel_Delta_Rx,
    kISpElementLabel_Delta_Yaw        = kISpElementLabel_Delta_Ry,
    kISpElementLabel_Delta_Cursor_X   = FOUR_CHAR_CODE('curx'),
    kISpElementLabel_Delta_Cursor_Y   = FOUR_CHAR_CODE('cury')
};
```

Constant Descriptions

`kISpElementLabel_Delta_X`
The element should adjust some x-axis delta value.

`kISpElementLabel_Delta_Y`
The element should adjust some y-axis delta value.

`kISpElementLabel_Delta_Z`
The element should adjust some z-axis delta value.

`kISpElementLabel_Delta_Rx`
The element should adjust rotation around the x-axis.

`kISpElementLabel_Delta_Ry`
The element should adjust rotation around the y-axis

`kISpElementLabel_Delta_Rz`
The element should adjust rotation around the z-axis

`kISpElementLabel_Delta_Roll`
The element should adjust the roll of a craft

`kISpElementLabel_Delta_Pitch`
The element should adjust the pitch of a craft.

`kISpElementLabel_Delta_Yaw`
The element should adjust the yaw of a craft.

`kISpElementLabel_Delta_Cursor_X`
: The element should adjust the x-axis position of a cursor.

`kISpElementLabel_Delta_Cursor_Y`
: The element should adjust the y-axis position of a cursor.

VERSION NOTES

These element-based labels introduced with InputSprocket 1.0.2.

Directional Pad Element Constants

You can use the following constants to indicate preferred usage for directional pad elements:

```
enum {
    kISpElementLabel_Pad_POV            = FOUR_CHAR_CODE('povh'),
    kISpElementLabel_Pad_Move           = FOUR_CHAR_CODE('move'),
    kISpElementLabel_Pad_POV_Horiz      = FOUR_CHAR_CODE('hpov'),
    kISpElementLabel_Pad_Move_Horiz     = FOUR_CHAR_CODE('hmov')
};
```

Constant Descriptions

`kISpElementLabel_Pad_POV`
: The element should control the player's point of view.

`kISpElementLabel_Pad_Move`
: The element should control the player's movement.

`kISpElementLabel_Pad_POV_Horiz`
: The element should control the player's point of view in a horizontal plane(that is, forward, backwards, left, and right).

`kISpElementLabel_Pad_Move_Horiz`
: The element should control the player's movement in a horizontal plane (that is, forward, backwards, left and right).

VERSION NOTES

These element-based labels introduced with InputSprocket 1.0.2.

Button Element Constants

You can use the following constants to indicate preferred usage for button elements:

```
enum {
    kISpElementLabel_Btn_Fire           = FOUR_CHAR_CODE('fire'),
    kISpElementLabel_Btn_SecondaryFire  = FOUR_CHAR_CODE('sfir'),
    kISpElementLabel_Btn_Jump           = FOUR_CHAR_CODE('jump'),
    kISpElementLabel_Btn_Quit           = FOUR_CHAR_CODE('strt'),
    kISpElementLabel_Btn_StartPause     = FOUR_CHAR_CODE('paus'),
    kISpElementLabel_Btn_Select         = FOUR_CHAR_CODE('optn'),
    kISpElementLabel_Btn_SlideLeft      = FOUR_CHAR_CODE('blft'),
    kISpElementLabel_Btn_SlideRight     = FOUR_CHAR_CODE('brgt'),
    kISpElementLabel_Btn_MoveForward    = FOUR_CHAR_CODE('btmf'),
    kISpElementLabel_Btn_MoveBackward   = FOUR_CHAR_CODE('btmb'),
    kISpElementLabel_Btn_TurnLeft       = FOUR_CHAR_CODE('bttl'),
    kISpElementLabel_Btn_TurnRight      = FOUR_CHAR_CODE('bttr'),
    kISpElementLabel_Btn_LookLeft       = FOUR_CHAR_CODE('btll'),
    kISpElementLabel_Btn_LookRight      = FOUR_CHAR_CODE('btlr'),
    kISpElementLabel_Btn_LookUp         = FOUR_CHAR_CODE('btlu'),
    kISpElementLabel_Btn_LookDown       = FOUR_CHAR_CODE('btld'),
    kISpElementLabel_Btn_Next           = FOUR_CHAR_CODE('btnx'),
    kISpElementLabel_Btn_Previous       = FOUR_CHAR_CODE('btpv'),
    kISpElementLabel_Btn_SideStep       = FOUR_CHAR_CODE('side'),
    kISpElementLabel_Btn_Run            = FOUR_CHAR_CODE('quik'),
    kISpElementLabel_Btn_Look           = FOUR_CHAR_CODE('blok'),
    kISpElementLabel_Btn_Minimum        = FOUR_CHAR_CODE('min '),
    kISpElementLabel_Btn_Decrement      = FOUR_CHAR_CODE('decr'),
    kISpElementLabel_Btn_Center         = FOUR_CHAR_CODE('cent'),
    kISpElementLabel_Btn_Increment      = FOUR_CHAR_CODE('incr'),
    kISpElementLabel_Btn_Maximum        = FOUR_CHAR_CODE('max '),
    kISpElementLabel_Btn_10Percent      = FOUR_CHAR_CODE(' 10 '),
    kISpElementLabel_Btn_20Percent      = FOUR_CHAR_CODE(' 20 '),
    kISpElementLabel_Btn_30Percent      = FOUR_CHAR_CODE(' 30 '),
    kISpElementLabel_Btn_40Percent      = FOUR_CHAR_CODE(' 40 '),
    kISpElementLabel_Btn_50Percent      = FOUR_CHAR_CODE(' 50 '),
    kISpElementLabel_Btn_60Percent      = FOUR_CHAR_CODE(' 60 '),
    kISpElementLabel_Btn_70Percent      = FOUR_CHAR_CODE(' 70 '),
    kISpElementLabel_Btn_80Percent      = FOUR_CHAR_CODE(' 80 '),
    kISpElementLabel_Btn_90Percent      = FOUR_CHAR_CODE(' 90 '),
    kISpElementLabel_Btn_MouseOne       = FOUR_CHAR_CODE('mou1'),
    kISpElementLabel_Btn_MouseTwo       = FOUR_CHAR_CODE('mou2'),
    kISpElementLabel_Btn_MouseThree     = FOUR_CHAR_CODE('mou3')
};
```

`kISpElementLabel_Btn_Fire`
: The element should be a fire button.

`kISpElementLabel_Btn_Secondary_Fire`
: The element should be a secondary fire button.

`kISpElementLabel_Btn_Jump`
: The element should be a jump button.

`kISpElementLabel_Btn_Quit`
: The button should quit the game. Note that this control is automatically associated with the Escape key if the keyboard is enabled.

`kISpElementLabel_Btn_StartPause`
: The button should pause the game or restart a paused game. Typically this button corresponds to the Start button on gamepads.

`kISpElementLabel_Btn_Select`
: The element should be a selection button (for example, to allow selection of a highlighted item in an inventory list).

`kISpElementLabel_Btn_SlideLeft`
: The button should let the player slide to the left.

`kISpElementLabel_Btn_SlideRight`
: The button should be let the player slide to the right.

`kISpElementLabel_Btn_MoveForward`
: The button should move the player move forward.

`kISpElementLabel_Btn_MoveBackward`
: The button should move the player backwards.

`kISpElementLabel_Btn_TurnLeft`
: The button should let the player turn left.

`kISpElementLabel_Btn_TurnRight`
: The button should let the player turn right.

`kISpElementLabel_Btn_Look Left`
: The button should let the player look left.

`kISpElementLabel_Btn_LookRight`
: The button should let the player look right.

`kISpElementLabel_Btn_LookUp`
: The button should let the player look up.

`kISpElementLabel_Btn_LookDown`
: The button should let the player look down.

`kISpElementLabel_Btn_Next`
: The button should select the next item in a list (such as an inventory or weapon list).

`kISpElementLabel_Btn_Previous`
: The button should select the previous item in a list (such as an inventory or weapon list).

`kISpElementLabel_Btn_SideStep`
: The button should let the player sidestep (used in conjunction with a directional control).

`kISpElementLabel_Btn_Run`
: The button should let the user run (often used in conjunction with a directional control).

`kISpElementLabel_Btn_Look`
: The button should let the player look (used in conjunction with a directional control).

`kISpElementLabel_Btn_Minimum`
: The button should return a setting to its minimum value.

`kISpElementLabel_Btn_Decrement`
: The button should decrement a setting.

`kISpElementLabel_Btn_Center`
: The button should return a control to its center position (for example, a steering rudder).

`kISpElementLabel_Btn_Increment`
: The button should increment a setting.

`kISpElementLabel_Btn_Maximum`
: The button should change a setting to its maximum value.

`kISpElementLabel_Btn_10Percent`
: The button should change a setting to 10% of its maximum value.

`kISpElementLabel_Btn_20Percent`
: The button should change a setting to 20% of its maximum value.

`kISpElementLabel_Btn_30Percent`
: The button should change a setting to 30% of its maximum value.

`kISpElementLabel_Btn_40Percent`
: The button should change a setting to 40% of its maximum value.

`kISpElementLabel_Btn_50Percent`
: The button should change a setting to 50% of its maximum value.

`kISpElementLabel_Btn_60Percent`
: The button should change a setting to 60% of its maximum value.

`kISpElementLabel_Btn_70Percent`
: The button should change a setting to 70% of its maximum value.

`kISpElementLabel_Btn_80Percent`
: The button should change a setting to 80% of its maximum value.

`kISpElementLabel_Btn_90Percent`
: The button should change a setting to 90% of its maximum value.

`kISpElementLabel_Btn_MouseOne`
: The button should correspond to the first button on a mouse.

`kISpElementLabel_Btn_MouseTwo`
: The button should correspond to the second button on a mouse.

`kISpElementLabel_Btn_MouseThree`
: The button should correspond to the third button on a mouse.

VERSION NOTES

These element-based labels introduced with InputSprocket 1.0.2.

Need Flags

These constants indicate specific attributes of needs, which you can set in the `flags` bit field of an `ISpNeed` structure.

```
typedef enum ISpNeedFlagBits {
    kISpNeedFlag_NoMultiConfig              = 0x00000001,
    kISpNeedFlag_Utility                    = 0x00000002,
    kISpNeedFlag_PolledOnly                 = 0x00000004,
    kISpNeedFlag_EventsOnly                 = 0x00000008,
    kISpNeedFlag_NoAutoConfig               = 0x00000010,
    kISpNeedFlag_NoConfig                   = 0x00000020,
    kISpNeedFlag_Invisible                  = 0x00000040,

/* *** kISpElementKind specific flags ****/
/* these are flags specific to kISpElementKind_Button*/

    kISpNeedFlag_Button_AlreadyAxis         = 0x10000000,
    kISpNeedFlag_Button_ClickToggles        = 0x20000000,
    kISpNeedFlag_Button_ActiveWhenDown      = 0x40000000,
    kISpNeedFlag_Button_AlreadyDelta        = (long)0x80000000,

/* these are flags specific to kISpElementKind_Axis*/

    kISpNeedFlag_Axis_AlreadyButton         = 0x10000000,
    kISpNeedFlag_Axis_Asymetric             = 0x20000000,
    kISpNeedFlag_Axis_AlreadyDelta          = 0x40000000,

/* these are flags specific to kISpElementKind_Delta*/

    kISpNeedFlag_Delta_AlreadyAxis          = 0x10000000,
    kISpNeedFlag_Delta_AlreadyButton        = 0x20000000
};
```

Constant description

`kISpNeedFlag_NoMultiConfig`
: Allows only one device to bind to this requirement during autoconfiguration.

`kISpNeedFlag_Utility`
: Indicates that the need is a utility function (such as volume,

screen resolution, or map) which would typically be assigned to a keyboard.

`kISpNeedFlag_PolledOnly`
Indicates that you can get information about this need only by polling it. InputSprocket will not allocate an event queue for the element associated with this need.

`kISpNeedFlag_EventsOnly`
Indicates that you can get information about this need only by checking for events.

`kISpNeedFlag_NoAutoConfig`
Indicates that autoconfiguration is not set. That is, this need will never show up as the default configuration choice (but the user can select it manually).

`kISpNeedFlag_NoConfig`
Indicates that the user cannot change the configuration of this need.

`kISpNeedFlag_Invisible`
Indicates that this need is not visible to the user. It will not show up on any autoconfiguration screens.

`kISpNeedFlag_Button_AlreadyAxis`
Indicates that an axis version of this button need also exists.

`kISpNeedFlag_Button_ClickToggles`
Indicates that a press of this button toggles between two states.

`kISpNeedFlag_Button_ActiveWhenDown`
Indicates that the need is activated only when the button is pushed.

`kISpNeedFlag_Button_AlreadyDelta`
Indicates that a delta version of this button also exists.

`kISpNeedFlag_Axis_AlreadyButton`
Indicates that a button version of this axis need also exists.

`kISpNeedFlag_Axis_Asymetric`
Indicates that this axis is asymmetric (that is, there is no logical center position).

`kISpNeedFlag_Axis_AlreadyDelta`
Indicates that a delta version of this axis need also exists.

`kISpNeedFlag_Delta_AlreadyAxis`
Indicates that an axis version of this delta need also exists.

`kISpNeedFlag_Delta_AlreadyButton`
Indicates that a button version of this delta need also exists.

VERSION NOTES

Introduced with InputSprocket 1.0.

Button Element Data Information

These constants specify information for data of kind `kButtonData`. Use the `ISpButtonConfigurationInfo` structure for help in interpreting the data.

```
typedef enum {
    kISpButtonUp =              0,
    kISpButtonDown =            1
} ISpButtonData;
```

Constant descriptions

`kISpButtonUp`	Button is up.
`kISpButtonDown`	Button is down.

VERSION NOTES

Introduced with InputSprocket 1.0.

Directional Pad Element Data Information

These constants specify information for data of kind `kElementData`. Use the `ISpDPadConfigurationInfo` structure for help in interpreting the data.

```
typedef enum {
    kISpPadIdle =               0,
    kISpPadLeft =               1,
    kISpPadUpLeft =             2,
    kISpPadUp =                 3,
    kISpPadUpRight =            4,
    kISpPadRight =              5,
    kISpPadDownRight =          6,
    kISpPadDown =               7,
    kISpPadDownLeft =           8
} ISpDPadData;
```

Constant descriptions

`kISpPadIdle`	Idle position.
`kISpPadLeft`	Left position.
`kISpPadUpLeft`	Upper-left position.
`kISpPadUp`	Upper-center position.
`kISpPadUpRight`	Upper-right position.
`kISpPadRight`	Right position.

VERSION NOTES

Introduced with InputSprocket 1.0.

Axis Element Data Information

These constants specify information for data of kind `kAxisData`. Use the `ISpAxisConfigurationInfo` structure for help in interpreting the data.

```
#define kISpAxisMinimum  0x00000000U
#define kISpAxisMiddle   0x7FFFFFFFU
#define kISpAxisMaximum  0xFFFFFFFFU
```

Constant descriptions

`kISpAxisMinimum`	Minimum distance along the axis.
`kISpAxisMiddle`	Center of the axis.
`kISpAxisMaximum`	Maximum distance along the axis.

VERSION NOTES

Introduced with InputSprocket 1.0.

Virtual Element Flag

Use this constant with the `ISpElement_NewVirtual` and `ISpElement_NewVirtualFromNeeds` functions to tell InputSprocket to allocate the virtual elements in temporary memory.

```
{
    kISpVirtualElementFlag_UseTempMem = 1
};
```

Constant descriptions

`kISpVirtualElementFlag_UseTempMem`
Allocate the virtual element in temporary memory.

VERSION NOTES

Introduced with InputSprocket 1.0.

Element List Flag

Use this constant with the `ISpElementList_New` function to tell InputSprocket to allocate the element list in temporary memory.

```
enum
{
    kISpElementListFlag_UseTempMem = 1
};
```

Constant descriptions

`kISpElementListFlag_UseTempMem`
Allocate the element list in temporary memory.

VERSION NOTES

Introduced with InputSprocket 1.0.

Device Emulation Flag

This constant indicates to InputSprocket that a device supports non-InputSprocket input schemes and that InputSprocket does not need to support it. Only device drivers have any need for this constant.

```
enum
{
    kISpDeviceFlag_HandleOwnEmulation = 1
};
```

Constant descriptions

`kISpDeviceFlag_HandleOwnEmulation`
: The device can handle a non-InputSprocket input scheme by itself.

VERSION NOTES

Introduced with InputSprocket **<<what?>>**

Icon Suite Constants

These constants define icons that developers may use in configuration screens. Currently InputSprocket defines only one icon.

```
enum {
    kISpFirstIconSuite      = 30000,
    kISpLastIconSuite       = 30100,
    kISpNoneIconSuite       = 30000
};
```

Constant descriptions

`kISpFirstIconSuite`
: Beginning of range for InputSprocket-defined icons.

`kISpLastIconSuite`
: End of range for InputSprocket-defined icons.

`kISpLNoneIconSuite`
: Icon to indicate that no need currently maps to that element.

VERSION NOTES

Introduced with InputSprocket **<<what?>>**

Resource Types

These constants identify resource types defined by InputSprocket. See "Resources" (page 78).

```
enum
{
    kISpApplicationResourceType =   'isap'
    kISpSetListResourceType =       'setl',
    kISpSetDataResourceType =       'tset'
};
```

Constant descriptions

`kISpApplicationResourceType`
: A resource of type `'isap'`.

`kISpSetListResourceType`
: A resource of type `'setl'`.

`kISpSetDataResourceType`
: A resource of type `'tset'`.

VERSION NOTES

Introduced with InputSprocket 1.0.

Application Resource Constants

You use these constants in the `ISpApplicationResourceStruct` structure to indicate properties of the application that calls InputSprocket.

```
enum {
    kISpAppResFlag_UsesInputSprocket    = 0x00000001,
    kISpAppResFlag_UsesISpInit          = 0x00000002
};
```

Constant descriptions

`kISpAppResFlag_UsesInputSprocket`
: Set this bit if the application calls InputSprocket.

`kISpAppResFlag_UsesISpInit`
: Set this bit if the application uses the high-level InputSprocket interface. That is, if it calls `ISpInit`, calls `ISpConfigure`, uses a needs list, and so on.

VERSION NOTES

Introduced with InputSprocket 1.2.

Resources

This section describes the following resources used with InputSprocket:

- "The Application Resource" (page 78)
- "The InputSprocket Set Resource" (page 79)
- "The InputSprocket Set List Resource" (page 79)

The Application Resource

The application resource, which is a resource of type `'isap'`, provides information about how an application uses InputSprocket. It consists of one 4-byte flag field and 3 reserved fields of 4 bytes each. See "Application Resource Constants" (page 77) for possible flag values.

Typically, a third-party device driver may want to examine this resource and adjust its actions depending on how the application uses InputSprocket.

VERSION NOTES

Introduced with InputSprocket 1.2.

The InputSprocket Set Resource

An InputSprocket set resource, a resource of type `'tset'`, is simply a block of data. The interpretation is up to the individual driver. Do not attempt to edit a set resource except through the dialog box provided by the `ISpConfigure` function. The `ISpConfigure` function saves each set resource in the InputSprocket preferences file. You should then copy the information in the preferences file to your game's resource file.

To get the best results from autoconfiguration you should have resources of this type inside your application.

VERSION NOTES

Introduced with InputSprocket 1.0.

The InputSprocket Set List Resource

You use a set list resource to specify a list of initial saved sets of preferences for the various devices that a user might use to play your game. You use this resource (as well as the set resource) to build a set list to pass to the `ISpInit` function. A set list resource allows you to have one default configuration as well as various other named configurations. A set list resource is of type `'setl'`.

The set list resource contains the following elements:

- Version. This 4-byte field specifies the version of the `'setl'` data type. The current version is `0x00000001`.
- Count. This 4-byte field specifies the number of set list entries that follow.
- Set list entries. Following the count field is an array of set list entry records. Each set list entry record occupies an 80-byte field. A single set list may have multiple sets in it. These sets may correspond to multiple devices as well as multiple configurations for a single device.

The set list entry record contains the following elements:

- Name. This 64-byte field contains the name of this set. It will be displayed to the user.
- Set length. This 4-byte field is the length of data in the `'tset'` resource that corresponds to this entry.

- Device identifier. This 4-byte field identifies the device that this set of entry records is for.
- Flags. Two fields are defined for the 4-byte flag field. All others are reserved and should be set to zero. For the first field you should OR in `kSetListEntryNotSelected`. The following information is provided for completeness only:

```
kSetListEntrySelectedMask = 0x000000001
kSetListEntryIsSelected =   0x000000001  // set list entry is selected
kSetListEntryNotSelected =  0x000000000  // set list entry is not
                                         // selected
```

For the second field you should OR in `kSetListEntryDefaultEntry` if you want this entry to be the default; otherwise, OR in `kSetListEntryAppEntry`.

```
kSetListEntryWhoMask =      0x000000006
kSetListEntryNormalEntry =  0x000000000     // a normal user-created
                                            // entry
kSetListEntryDefaultEntry = 0x000000002     // the default entry
kSetListEntryAppEntry =     0x000000004     // an application-provided
                                            // entry
kSetListEntryDriverEntry =  0x000000006     // a driver-provided entry
```

- Set resource ID. This 2-byte field contains the resource ID of the `'tset'` resource that corresponds to this entry.
- Reserved. This 2-byte field is reserved and should be set to zero.

VERSION NOTES

Introduced with InputSprocket 1.0.

Summary of InputSprocket

Functions

Invoking and Configuring InputSprocket

```
OSStatus ISpInit                        (UInt32 count,
                                         ISpNeed *needs,
                                         ISpElementReference *inReferences,
                                         OSType appCreatorCode,
                                         OSType subCreatorCode,
                                         UInt32 flags,
                                         short setListResourceID
                                         UInt32 reserved);

OSStatus ISpConfigure                   (ISpEventProcPtr inEventProcPtr);

OSStatus ISpStartup (void);

OSStatus ISpShutdown (void);

OSStatus ISpStop                        (void);

OSStatus ISpSuspend                     (void);

OSStatus ISpResume                      (void);

OSStatus ISpTickle (void);

NumVersion ISpGetVersion                (void);
```

Manipulating Elements

```
OSStatus ISpElement_NewVirtual          (UInt32 dataSize,
                                         ISpElementReference *outElement;
                                         UInt32 flags);
```

```
OSStatus ISpElement_NewVirtualFromNeeds (UInt32 count,
                                         ISpNeed *needs,
                                         ISpElementReference *outElements
                                         UInt32 flags);

OSStatus ISpElement_DisposeVirtual      (UInt32 count,
                                         ISpElementReference
                                         *inElements);

OSStatus ISpElement_GetDevice           (const ISpElementReference inElement,
                                         ISpDeviceReference *outDevice);

OSStatus ISpElement_GetGroup            (const ISpElementReference inElement,
                                         UInt32 *outGroup);

OSStatus ISpElement_GetInfo             (const ISpElementReference inElement,
                                         ISpElementInfoPtr outInfo);

OSStatus ISpElement_GetConfigurationInfo(const ISpElementReference inElement,
                                         UInt32 buflen, void *configInfo);
```

Obtaining Data From Elements

```
OSStatus ISpElement_GetSimpleState      (const ISpElementReference inElement,
                                         Uint32 *state);

OSStatus ISpElement_GetComplexState     (const ISpElementReference inElement,
                                         UInt32 buflen, void *state);

OSStatus ISpElement_GetNextEvent        (ElemenReference inElement,
                                         UInt32 bufSize,
                                         ISpElementEventPtr event,
                                         Boolean *wasEvent);

OSStatus ISpElementList_GetNextEvent    (ISpElementListReference inElementList,
                                         UInt32 bufSize,
                                         ISpElementEventPtr event,
                                         Boolean *wasEvent);

AbsoluteTime ISpUptime                  (void);

OSStatus ISpTimeToMicroseconds          (const AbsoluteTime *inTime,
                                         UnsignedWide *outMicroseconds);

OSStatus ISpElement_Flush               (ISpElementReference inElement);
OSStatus ISpElementList_Flush          (ISpElementListReference inElementList);
```

Manipulating Devices

```
OSStatus ISpDevices_Extract            (UInt32 inBufferCount,
                                        UInt32 *outCount,
                                        ISpDeviceReference *buffer);

OSStatus ISpDevices_ExtractByClass     (ISpDeviceClass theClass,
                                        UInt32 inBufferCount,
                                        UInt32 *outCount,
                                        ISpDeviceReference *buffer);

OSStatus ISpDevices_ExtractByIdentifier (ISpDeviceIdentifier theIdentifier,
                                        UInt32 inBufferCount,
                                        UInt32 *outCount,
                                        ISpDeviceReference *buffer);

OSStatus ISpDevices_Activate           (UInt32 inDeviceCount,
                                        ISpDeviceReference *inDevicesToActivate);

OSStatus ISpDevices_Deactivate         (UInt32 inDeviceCount,
                                        ISpDeviceReference *inDevicesToDeactivate);

OSStatus ISpDevices_ActivateClass      (ISpDeviceClass inClass);

OSStatus ISpDevices_DeactivateClass    (ISpDeviceClass inClass);

OSStatus ISpDevice_IsActive            (ISpDeviceReference inDevice,
                                        Boolean *outIsActive);

OSStatus ISpDevice_GetDefinition       (const ISpDeviceReference inDevice,
                                        UInt32 buflen,
                                        ISpDeviceDefinition *outStruct);

OSStatus ISpDevice_GetElementList      (const ISpDeviceReference inDevice,
                                        ISpElementListReference *outElementList);
```

Managing Element Lists

```
OSStatus ISpElementList_New            (UInt32 inCount,
                                        ISpElementReference *inElements,
                                        ISpElementListReference *outElementList
                                        UInt32 flags);
```

```
OSStatus ISpElementList_Dispose        (ISpElementListReference inElementList);

OSStatus ISpGetGlobalElementList       (ISpElementListReference *outElementList);

OSStatus ISpElementList_AddElements    (ISpElementListReference inElementList,
                                        UInt32 refCon,
                                        UInt32 count,
                                        ISpElementReference *newElements);

OSStatus ISpElementList_RemoveElements (ISpElementListReference inElementList,
                                        UInt32 count,
                                        ISpElementReference *oldElement);

OSStatus ISpElementList_Extract        (ISpElementListReference inElementList,
                                        UInt32 inBufferCount,
                                        UInt32 *outCount,
                                        ISpElementReference *buffer);

OSStatus ISpElementList_ExtractByKind  (ISpElementListReference inElementList,
                                        ISpElementKind theKind,
                                        UInt32 inBufferCount,
                                        UInt32 *outCount,
                                        ISpElementReference *buffer);

OSStatus ISpElementList_ExtractByLabel (ISpElementListReference inElementList,
                                        ISpElementLabel theLabel,
                                        UInt32 inBufferCount,
                                        UInt32 *outCount,
                                        ISpElementReference *buffer);
```

Application-Defined Function

```
Boolean MyEventProc                    (EventRecord *inEvent);
```

Data Types

```
typedef struct OpaqueISpDeviceReference     *ISpDeviceReference;

typedef struct OpaqueISpElementReference    *ISpElementReference;

typedef struct OpaqueISpElementListReference *ISpElementListReference;
typedef OSType ISpDeviceClass;

typedef OSType ISpDeviceIdentifier;

typedef OSType ISpElementLabel;

typedef OSType ISpElementKind;
```

Need Structure

```
typedef struct ISpNeed {
    Str63                   name;
    short                   iconSuiteResourceId;
    UInt8                   playerNum
    UInt8                   group
    ISpElementKind          theKind;
    ISpElementLabel         theLabel;
    UInt32                  flags;
    UInt32                  reserved1;
    UInt32                  reserved2;
    UInt32                  reserved3;
} ISpNeed;
```

Event Handler Procedure Definition

```
typedef Boolean (*ISpEventProcPtr)(EventRecord *inEvent);
```

Element Event Data Structure

```
typedef struct {
    AbsoluteTime            when;
    ISpElementReference     element;
    UInt32                  refCon;
    Uint32                  data;
} ISpElementEvent, *ISpElementEventPtr;
```

Device Definition Structure

```
typedef struct ISpDeviceDefinition {
    Str63                   deviceName;
    ISpDeviceClass          theDeviceClass;
    ISpDeviceIdentifier     theDeviceIdentifier;
    UInt32                  permanentId;
    UInt32                  flags;
    UInt32                  permanentIDExtend;
    UInt32                  reserved2;
    UInt32                  reserved3;
} ISpDeviceDefinition;
```

Element Information Structure

```
typedef struct ISpElementInfo {
    ISpElementLabel             theLabel;
    ISpElementKind              theKind;
    Str63                       theString;
    UInt32                      reserved1;
    UInt32                      reserved2;
} ISpElementInfo, *ISpElementInfoPtr;
```

Button Configuration Information Structure

```
typedef struct {
    UInt32                      id;
} ISpButtonConfigurationInfo;
```

Directional Pad Configuration Information Structure

```
typedef struct {
    UInt32                      id;
    Boolean                     fourWayPad;
} ISpDPadConfigurationInfo;
```

Axis Configuration Information Structure

```
typedef struct {
    Boolean                     SymetricAxis;
} ISpAxisConfigurationInfo;
```

Delta Configuration Information Structure

```
typedef struct ISpDeltaConfigurationInfo {
    UInt32 reserved1;
    UInt32 reserved2;
    } ISpDeltaConfigurationInfo;
```

Movement Configuration Information Structure

```
typedef struct ISpMovementConfigurationInfo {
    UInt32 reserved1;
    UInt32 reserved2;
    }; ISpMovementConfigurationInfo;
```

Movement Data Structure

```
typedef struct ISpMovementData {
    UInt32                      xAxis;
    UInt32                      yAxis;
    UInt32                      direction;
};
```

Application Resource Structure

```
typedef struct ISpApplicationResourceStruct {
    UInt32                      flags;
    UInt32                      reserved1;
    UInt32                      reserved2;
    UInt32                      reserved3;
}; ISpApplicationResourceStruct;
```

Constants

Built-in Device Categories

```
enum {
    kISpDeviceClass_SpeechRecognition       = FOUR_CHAR_CODE('talk'),
    kISpDeviceClass_Mouse                   = FOUR_CHAR_CODE('mous'),
    kISpDeviceClass_Keyboard                = FOUR_CHAR_CODE('keyd'),
    kISpDeviceClass_Joystick                = FOUR_CHAR_CODE('joys'),
    kISpDeviceClass_Gamepad                 = FOUR_CHAR_CODE('gmpd'),
    kISpDeviceClass_Wheel                   = FOUR_CHAR_CODE('whel'),
    kISpDeviceClass_Pedals                  = FOUR_CHAR_CODE('pedl'),
    kISpDeviceClass_Levers                  = FOUR_CHAR_CODE('levr'),
    kISpDeviceClass_Tickle                  = FOUR_CHAR_CODE('tckl'),
    kISpDeviceClass_Unknown                 = FOUR_CHAR_CODE('????')
};
```

Built-in Element Kinds

```
enum {
    kISpElementKind_Button      = FOUR_CHAR_CODE('butn'),
    kISpElementKind_DPad        = FOUR_CHAR_CODE('dpad'),
    kISpElementKind_Axis        = FOUR_CHAR_CODE('axis'),
    kISpElementKind_Delta       = FOUR_CHAR_CODE('dlta'),
    kISpElementKind_Movement    = FOUR_CHAR_CODE('move'),
    kISpElementKind_Virtual     = FOUR_CHAR_CODE('virt')
};
```

Element Label Constants

```
enum {
    /* generic */
    kISpElementLabel_None               = FOUR_CHAR_CODE('none'),

    /* axis */
    kISpElementLabel_Axis_XAxis         = FOUR_CHAR_CODE('xaxi'),
    kISpElementLabel_Axis_YAxis         = FOUR_CHAR_CODE('yaxi'),
    kISpElementLabel_Axis_ZAxis         = FOUR_CHAR_CODE('zaxi'),
    kISpElementLabel_Axis_Rx            = FOUR_CHAR_CODE('rxax'),
    kISpElementLabel_Axis_Ry            = FOUR_CHAR_CODE('ryax'),
    kISpElementLabel_Axis_Rz            = FOUR_CHAR_CODE('rzax'),
    kISpElementLabel_Axis_Roll          = kISpElementLabel_Axis_Rz,
    kISpElementLabel_Axis_Pitch         = kISpElementLabel_Axis_Rx,
    kISpElementLabel_Axis_Yaw           = kISpElementLabel_Axis_Ry,
    kISpElementLabel_Axis_RollTrim      = FOUR_CHAR_CODE('rxtm'),
    kISpElementLabel_Axis_PitchTrim     = FOUR_CHAR_CODE('trim'),
    kISpElementLabel_Axis_YawTrim       = FOUR_CHAR_CODE('rytm'),
    kISpElementLabel_Axis_Gas           = FOUR_CHAR_CODE('gasp'),
    kISpElementLabel_Axis_Brake          = FOUR_CHAR_CODE('brak'),
    kISpElementLabel_Axis_Clutch         = FOUR_CHAR_CODE('cltc'),
    kISpElementLabel_Axis_Throttle       = FOUR_CHAR_CODE('thrt'),
    kISpElementLabel_Axis_Trim           = kISpElementLabel_Axis_PitchTrim,
    kISpElementLabel_Axis_Rudder         = FOUR_CHAR_CODE('rudd'),
    kISpElementLabel_Axis_ToeBrake       = FOUR_CHAR_CODE('toeb'),

    /* delta */
    kISpElementLabel_Delta_X             = FOUR_CHAR_CODE('xdlt'),
    kISpElementLabel_Delta_Y             = FOUR_CHAR_CODE('ydlt'),
    kISpElementLabel_Delta_Z             = FOUR_CHAR_CODE('zdlt'),
    kISpElementLabel_Delta_Rx            = FOUR_CHAR_CODE('rxdl'),
    kISpElementLabel_Delta_Ry            = FOUR_CHAR_CODE('rydl'),
    kISpElementLabel_Delta_Rz            = FOUR_CHAR_CODE('rzdl'),
    kISpElementLabel_Delta_Roll          = kISpElementLabel_Delta_Rz,
    kISpElementLabel_Delta_Pitch         = kISpElementLabel_Delta_Rx,
    kISpElementLabel_Delta_Yaw           = kISpElementLabel_Delta_Ry,
    kISpElementLabel_Delta_Cursor_X      = FOUR_CHAR_CODE('curx'),
    kISpElementLabel_Delta_Cursor_Y      = FOUR_CHAR_CODE('cury'),

    /* direction pad */
    kISpElementLabel_Pad_POV             = FOUR_CHAR_CODE('povh'),
    kISpElementLabel_Pad_Move            = FOUR_CHAR_CODE('move'),
    kISpElementLabel_Pad_POV_Horiz       = FOUR_CHAR_CODE('hpov'),
    kISpElementLabel_Pad_Move_Horiz      = FOUR_CHAR_CODE('hmov'),

    /* buttons */
    kISpElementLabel_Btn_Fire            = FOUR_CHAR_CODE('fire'),
    kISpElementLabel_Btn_SecondaryFire   = FOUR_CHAR_CODE('sfir'),
    kISpElementLabel_Btn_Jump            = FOUR_CHAR_CODE('jump'),
    kISpElementLabel_Btn_Quit            = FOUR_CHAR_CODE('strt'),
    kISpElementLabel_Btn_StartPause      = FOUR_CHAR_CODE('paus'),
    kISpElementLabel_Btn_Select          = FOUR_CHAR_CODE('optn'),
    kISpElementLabel_Btn_SlideLeft       = FOUR_CHAR_CODE('blft'),
    kISpElementLabel_Btn_SlideRight      = FOUR_CHAR_CODE('brgt'),
    kISpElementLabel_Btn_MoveForward     = FOUR_CHAR_CODE('btmf'),
    kISpElementLabel_Btn_MoveBackward    = FOUR_CHAR_CODE('btmb'),
    kISpElementLabel_Btn_TurnLeft        = FOUR_CHAR_CODE('bttl'),
    kISpElementLabel_Btn_TurnRight       = FOUR_CHAR_CODE('bttr'),
    kISpElementLabel_Btn_LookLeft        = FOUR_CHAR_CODE('btll'),
    kISpElementLabel_Btn_LookRight       = FOUR_CHAR_CODE('btlr'),
    kISpElementLabel_Btn_LookUp         = FOUR_CHAR_CODE('btlu'),
    kISpElementLabel_Btn_LookDown       = FOUR_CHAR_CODE('btld'),
    kISpElementLabel_Btn_Next           = FOUR_CHAR_CODE('btnx'),
    kISpElementLabel_Btn_Previous       = FOUR_CHAR_CODE('btpv'),
    kISpElementLabel_Btn_SideStep       = FOUR_CHAR_CODE('side'),
    kISpElementLabel_Btn_Run            = FOUR_CHAR_CODE('quik'),
    kISpElementLabel_Btn_Look           = FOUR_CHAR_CODE('blok'),
    kISpElementLabel_Btn_Minimum        = FOUR_CHAR_CODE('min '),
    kISpElementLabel_Btn_Decrement      = FOUR_CHAR_CODE('decr'),
    kISpElementLabel_Btn_Center         = FOUR_CHAR_CODE('cent'),
    kISpElementLabel_Btn_Increment      = FOUR_CHAR_CODE('incr'),
    kISpElementLabel_Btn_Maximum        = FOUR_CHAR_CODE('max '),
    kISpElementLabel_Btn_10Percent      = FOUR_CHAR_CODE(' 10 '),
    kISpElementLabel_Btn_20Percent      = FOUR_CHAR_CODE(' 20 '),
    kISpElementLabel_Btn_30Percent      = FOUR_CHAR_CODE(' 30 '),
    kISpElementLabel_Btn_40Percent      = FOUR_CHAR_CODE(' 40 '),
    kISpElementLabel_Btn_50Percent      = FOUR_CHAR_CODE(' 50 '),
    kISpElementLabel_Btn_60Percent      = FOUR_CHAR_CODE(' 60 '),
    kISpElementLabel_Btn_70Percent      = FOUR_CHAR_CODE(' 70 '),
    kISpElementLabel_Btn_80Percent      = FOUR_CHAR_CODE(' 80 '),
    kISpElementLabel_Btn_90Percent      = FOUR_CHAR_CODE(' 90 '),
    kISpElementLabel_Btn_MouseOne       = FOUR_CHAR_CODE('mou1'),
    kISpElementLabel_Btn_MouseTwo       = FOUR_CHAR_CODE('mou2'),
    kISpElementLabel_Btn_MouseThree     = FOUR_CHAR_CODE('mou3')
};
```

Need Flags

```
typedef enum ISpNeedFlagBits {
    kISpNeedFlag_NoMultiConfig              = 0x00000001,
    kISpNeedFlag_Utility                    = 0x00000002,
    kISpNeedFlag_PolledOnly                 = 0x00000004,
    kISpNeedFlag_EventsOnly                 = 0x00000008,
    kISpNeedFlag_NoAutoConfig               = 0x00000010,
    kISpNeedFlag_NoConfig                   = 0x00000020,
    kISpNeedFlag_Invisible                  = 0x00000040,

/* *** kISpElementKind specific flags ****/
/* these are flags specific to kISpElementKind_Button*/

    kISpNeedFlag_Button_AlreadyAxis         = 0x10000000,
    kISpNeedFlag_Button_ClickToggles        = 0x20000000,
    kISpNeedFlag_Button_ActiveWhenDown      = 0x40000000,
    kISpNeedFlag_Button_AlreadyDelta        = (long)0x80000000,

/* these are flags specific to kISpElementKind_Axis*/

    kISpNeedFlag_Axis_AlreadyButton         = 0x10000000,
    kISpNeedFlag_Axis_Asymetric             = 0x20000000,
    kISpNeedFlag_Axis_AlreadyDelta          = 0x40000000,

/* these are flags specific to kISpElementKind_Delta*/

    kISpNeedFlag_Delta_AlreadyAxis          = 0x10000000,
    kISpNeedFlag_Delta_AlreadyButton        = 0x20000000
};
```

Button Element Data Information

```
typedef enum {
    kISpButtonUp =                      0,
    kISpButtonDown =                    1
} ISpButtonData;
```

Directional Pad Element Data Information

```
typedef enum {
    kISpPadIdle =                    0,
    kISpPadLeft,
    kISpPadUpLeft,
    kISpPadUp,
    kISpPadUpRight,
    kISpPadRight,
    kISpPadDownRight,
    kISpPadDown,
    kISpPadDownLeft
} ISpDPadData;
```

Axis Element Data Information

```
#define kISpAxisMinimum  0x00000000U
#define kISpAxisMiddle   0x7FFFFFFFU
#define kISpAxisMaximum  0xFFFFFFFFU
```

Virtual Element Flag

```
enum
{
    kISpVirtualElementFlag_UseTempMem =  1
};
```

Element List Flag

```
enum
{
    kISpElementListFlag_UseTempMem =    1
};
```

Device Emulation Flag

```
enum
{
    kISpDeviceFlag_HandleOwnEmulation = 1
};
```

Icon Suite Constants

```
enum {
    kISpFirstIconSuite      = 30000,
    kISpLastIconSuite       = 30100,
    kISpNoneIconSuite       = 30000
};
```

Resource Types

```
enum
{
    kISpApplicationResourceType         = 'isap'
    kISpSetListResourceType             = 'setl',
    kISpSetDataResourceType             = 'setd'
};
```

Application Resource Constants

```
enum {
    kISpAppResFlag_UsesInputSprocket    = 0x00000001,
    kISpAppResFlag_UsesISpInit          = 0x00000002
};
```

Result Codes

`kISpInternalErr`	–30420	Internal error.
`kISpSystemListErr`	–30421	Operation is not allowed a system-maintained element list.
`kISpBufferToSmallErr`	–30422	The buffer is too small.
`kISpElementInListErr`	–30423	The element is already in the element list.
`kISpElementNotInListErr`	–30424	The element is not in the element list.
`kISpSystemInactiveErr`	–30425	InputSprocket is currently inactive.
`kISpDeviceInactiveErr`	–30426	The device is currently inactive.
`kISpSystemActiveErr`	–30427	Input Sprocket is currently active.
`kISpDeviceActiveErr`	–30428	The device is currently active.
`kISpListBusyErr`	–30429	The element list is marked as busy.

Document Version History

This document has had the following releases:

Table A-1 InputSprocket documentation revision history

Version	Notes
October 20, 1999	First preliminary release. This document reflects the changes to InputSprocket since version 1.0 documented in Chapter 3 of the Apple Game Sprockets Guide. A summary of changes is as follows: New functions added: `ISpStartup` (page 11), `ISpTickle` (page 16), `ISpShutdown` (page 16), `ISpDevices_ActivateClass` (page 34), `ISpDevices_DeactivateClass` (page 35), `ISpUptime` (page 26) and `ISpTimeToMicroseconds` (page 27). `ISpNeed` (page 48) structure modified to include player and group information. You should update your code to use this newer structure. New device categories added: levers, tickle, and unknown. See “Built-in Device Categories” (page 59). New delta element type added. See “Built-in Element Kinds” (page 61) and `ISpDeltaConfigurationInfo` (page 56). New element label constants defined. See “Element Label Constants” (page 62). You should use these constants in place of the older definitions. Additional need flags defined. See “Need Flags” (page 71). Device Emulation flag added. See “Device Emulation Flag” (page 76). Icon Suite constants defined. See “Icon Suite Constants” (page 76). New application resource defined. See “The Application Resource” (page 78) and “Application Resource Constants” (page 77).

A P P E N D I X

Document Version History

Index

T H E   A P P L E   P U B L I S H I N G   S Y S T E M

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITER
Jun Suzuki

Special thanks to Brent Schorsch, Chris DeSalvo, and Geoff Stahl, as well as Jasjeet Thind.

Acknowledgements to Dave Bice, Judy Helfland, Tim Monroe, and Larry Wood, who wrote the previous Game Sprockets guide.