Apple Game Sprockets Legacy Reference

(Legacy)

Mac OS 9 & Earlier > Unsupported

2003-01-15

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Apple Game Sprockets Legacy Reference (Legacy)

Framework: DrawSprocket/DrawSprocket.h
Declared in DrawSprocket.h

Important: The information in this document is obsolete and should not be used for new development.

Overview

This document describes Game Sprockets APIs that are no longer being supported. If you want to implement similar functionality on Mac OS X, you should investigate Mac OS X–specific APIs.

Functions by Task

Blitting Functions

`DSpBlit_Faster` (page 29)
Performs the specified blitting operation (including scaling).

`DSpBlit_Fastest` (page 29)
Performs the specified blitting operation (without scaling).

Choosing a Context

`DSpCanUserSelectContext` (page 30)
Determines whether there is a meaningful choice of contexts to present to the user with the `DSpUserSelectContext` function.

`DSpUserSelectContext` (page 38)
Presents a dialog box that allows the user to select a display.

Drawing and Double Buffering

`DSpContext_GetDirtyRectGridSize` (page 31)
Finds out the current grid size for a context's dirty rectangles.

`DSpContext_GetDirtyRectGridUnits` (page 32)
Finds out the size of the base dirty rectangle grid for a context.

`DSpContext_GetMaxFrameRate` (page 33)
Obtains the maximum frame rate for a specified context.

`DSpContext_InvalBackBufferRect` (page 34)
Invalidates a specific area of a context's back buffer, so that only a portion of the screen needs to be redrawn when the buffers are next swapped.

`DSpContext_SetDirtyRectGridSize` (page 36)
Suggests a grid size for the context's dirty rectangles.

`DSpContext_SetMaxFrameRate` (page 36)
Sets a maximum frame rate for a specified context.

Hosting and Joining a Game

`NSpGame_Dispose` (page 78)
Removes a player or host from the game.

`NSpGame_EnableAdvertising` (page 79)
Enables or disables advertising of the game on the network.

`NSpGame_GetInfo` (page 80)
Obtains information about an available game.

`NSpGame_Host` (page 80)
Creates a new game object that other players can then join.

`NSpGame_Join` (page 82)
Joins a game specified by an address.

`NSpInstallJoinRequestHandler` (page 90)
Installs the application-defined join request handler.

Human Interface Functions

`NSpDoModalHostDialog` (page 76)
Presents your user with a default modal dialog box for hosting a game on the network.

`NSpDoModalJoinDialog` (page 77)
Presents to the user a default dialog box for finding and joining a game advertised on the network.

Initializing NetSprocket

`NSpInitialize` (page 88)
Initializes the NetSprocket library.

Invoking and Configuring InputSprocket

`ISpConfigure` (page 42)
Uses the high-level InputSprocket layer to generate a modal window where the user can match device elements with the game's input requirements.

`ISpGetVersion` (page 67)
Returns the version of InputSprocket installed.

`ISpInit` (page 67)
Initializes the high-level InputSprocket layer and autoconfigures all the devices.

`ISpResume` (page 70)
Resumes InputSprocket.

`ISpShutdown` (page 71)
Shuts down InputSprocket and unloads all InputSprocket drivers.

`ISpStartup` (page 71)
Starts up InputSprocket and loads all the InputSprocket drivers.

`ISpStop` (page 71)
Stops the flow of data into the virtual elements and disposes of elements allocated by the `ISpInit` call.

`ISpSuspend` (page 72)
Suspends InputSprocket.

`ISpTickle` (page 72)
Allows InputSprocket to give up time to other InputSprocket drivers.

Managing Element Lists

`ISpElementList_AddElements` (page 51)
Adds more elements to an element list.

`ISpElementList_Dispose` (page 52)
Disposes of a specified element list.

`ISpElementList_Extract` (page 52)
Extracts and counts elements in an element list.

`ISpElementList_ExtractByKind` (page 53)
Extracts and counts elements of a specified kind in an element list.

`ISpElementList_ExtractByLabel` (page 54)
Extracts and counts elements with a specific label in an element list.

`ISpElementList_New` (page 56)
Creates a new element list.

`ISpElementList_RemoveElements` (page 57)
Removes elements from an element list.

`ISpGetGlobalElementList` (page 66)
Obtains a global element list, which is a list of all the elements in the system.

Managing Groups of Players

`NSpGroup_AddPlayer` (page 84)
Adds a player to a group.

`NSpGroup_Dispose` (page 84)
Removes a group from the game.

`NSpGroup_GetEnumeration` (page 85)
Obtains a list of the groups in the game.

`NSpGroup_GetInfo` (page 85)
Obtains the group's information structure.

`NSpGroup_New` (page 86)
Creates a new group of players.

`NSpGroup_ReleaseEnumeration` (page 86)
Releases memory held by the group enumeration structure.

`NSpGroup_ReleaseInfo` (page 87)
Releases memory held by the group information structure.

`NSpGroup_RemovePlayer` (page 87)
Removes a player from a group.

Managing Network Protocols

`NSpProtocol_CreateAppleTalk` (page 103)
Creates an AppleTalk protocol reference using the specified parameters.

`NSpProtocol_CreateIP` (page 104)
Creates an IP protocol reference.

`NSpProtocol_Dispose` (page 105)
Deletes a protocol reference.

`NSpProtocolList_Append` (page 100)
Adds a new protocol reference to the list.

`NSpProtocolList_Dispose` (page 100)
Deletes a protocol list.

`NSpProtocolList_GetCount` (page 101)
Returns the number of protocol references in the list.

`NSpProtocolList_GetIndexedRef` (page 101)
Receives the protocol reference at the indicated location in the list.

`NSpProtocolList_New` (page 102)
Creates a new list for storing multiple protocol references.

`NSpProtocolList_Remove` (page 102)
Removes a protocol reference from the list.

`NSpProtocolList_RemoveIndexed` (page 103)
Removes the protocol reference at a specific location in the list.

Managing Player Information

`NSpPlayer_ChangeType` (page 94)
Changes the player's type.

`NSpPlayer_GetAddress` (page 95)
Obtains a player's network address.

`NSpPlayer_GetEnumeration` (page 95)
Takes a snapshot that describes each player currently in the game.

`NSpPlayer_GetInfo` (page 96)
Obtains information about a player.

`NSpPlayer_GetMyID` (page 97)
Obtains the ID of the player associated with the game object on the current computer.

`NSpPlayer_GetThruput` (page 97)
Determines the data throughput between the caller and the specified player.

`NSpPlayer_ReleaseEnumeration` (page 98)
Releases the player enumeration structure.

`NSpPlayer_ReleaseInfo` (page 99)
Releases a player information structure obtained by the `NSpPlayer_GetInfo` function.

`NSpPlayer_Remove` (page 99)
Removes a player.

Manipulating Devices

`ISpDevice_GetDefinition` (page 47)
Obtains a device definition structure for a specified device.

`ISpDevice_GetElementList` (page 48)
Obtains an element list for a specified device.

`ISpDevice_IsActive` (page 48)
Finds out if a device is active.

`ISpDevices_Activate` (page 42)
Activates the specified devices.

`ISpDevices_ActivateClass` (page 43)
Activates the specified class of devices.

`ISpDevices_Deactivate` (page 44)
Deactivates the specified devices.

`ISpDevices_DeactivateClass` (page 44)
Deactivates the specified class of devices.

`ISpDevices_Extract` (page 45)
Extracts and counts devices listed on the systemwide list of devices.

`ISpDevices_ExtractByClass` (page 45)
Extracts and counts devices of a certain class listed on the systemwide list of devices.

`ISpDevices_ExtractByIdentifier` (page 46)
Extracts and counts devices of a certain type that are listed on the systemwide list of devices.

Manipulating Elements

`ISpElement_DisposeVirtual` (page 58)
Disposes of virtual elements.

`ISpElement_GetConfigurationInfo` (page 60)
Obtains configuration information for an element.

`ISpElement_GetDevice` (page 60)
Finds out what device an element belongs to.

`ISpElement_GetGroup` (page 61)
Finds out what group an element belongs to.

`ISpElement_GetInfo` (page 62)
Obtains an element information structure for an element.

`ISpElement_NewVirtual` (page 64)
Creates a single virtual element for an element with data of a certain size.

`ISpElement_NewVirtualFromNeeds` (page 64)
Allocates virtual elements for all items in a need structure array.

Obtaining Data From Elements

`ISpElement_Flush` (page 59)
Flushes the event queue associated with an element.

`ISpElement_GetComplexState` (page 59)
Obtains the current state of an element.

`ISpElement_GetNextEvent` (page 62)
Obtains event data for a single element.

`ISpElement_GetSimpleState` (page 63)
Obtains the current state of an element whose data fits in an unsigned 32-bit integer.

`ISpElementList_Flush` (page 55)
Flushes the event queue associated with an element list.

`ISpElementList_GetNextEvent` (page 56)
Obtains the most recent event from a list of elements.

`ISpTimeToMicroseconds` (page 73)
Converts from units of `AbsoluteTime` (as received in an InputSprocket event structure) to units of microseconds.

`ISpUptime` (page 73)
Obtains the time elapsed since machine startup.

Saving and Restoring a Context

`DSpContext_Flatten` (page 30)
Converts a context into a format suitable for saving to disk—for example, to save user preferences.

`DSpContext_GetFlattenedSize` (page 32)
Determines how much memory is required to store a flattened version of a context.

`DSpContext_Restore` (page 35)
Restores a context that was saved previously, most likely to preserve a user's preferences.

Sending and Receiving Messages

`NSpInstallAsyncMessageHandler` (page 89)
Installs a message handler for your game object.

`NSpMessage_Get` (page 91)
Receives messages that have been delivered to your game.

`NSpMessage_Release` (page 92)
Releases a message obtained by calling `NSpMessage_Get`.

`NSpMessage_Send` (page 92)
Delivers a message to other players in the game.

`NSpMessage_SendTo` (page 93)
Creates a message header and sends a message to other players in the game.

Using Alternate Buffers

`DSpAltBuffer_Dispose` (page 26)
Disposes of an alternate buffer.

`DSpAltBuffer_GetCGrafPtr` (page 26)
Obtains the drawing area for an alternate buffer.

`DSpAltBuffer_InvalRect` (page 27)
Invalidates a rectangle in an alternate buffer.

`DSpAltBuffer_New` (page 28)
Creates an alternate buffer for an underlay.

`DSpContext_GetUnderlayAltBuffer` (page 34)
Obtains the current underlay associated with a context.

`DSpContext_SetUnderlayAltBuffer` (page 37)
Designates an alternate buffer to be used as the current underlay buffer for a context.

Utility Functions

`DSpContext_SetVBLProc` (page 38)
Piggybacks your own VBL task to a particular context.

`NSpClearMessageHeader` (page 74)
Initializes the entire header structure.

`NSpConvertAddressReferenceToOTAddr` (page 75)
Obtains an Open Transport `OTAddress` from a NetSprocket `NSpAddressReference`.

`NSpConvertOTAddrToAddressReference` (page 75)
Obtains a NetSprocket `NSpAddressReference` from an Open Transport `OTAddress`.

`NSpGetCurrentTimeStamp` (page 83)
Compares time stamps.

`NSpGetVersion` (page 83)
Returns the version of NetSprocket.

`NSpReleaseAddressReference` (page 106)
Releases memory associated with an address reference allocated by NetSprocket.

`NSpSetConnectTimeout` (page 107)
Sets the timeout period to create a new network connection.

Miscellaneous

`DisposeDSpCallbackUPP` (page 25)

`DisposeDSpEventUPP` (page 25)

`GSpConfigure` (page 39)

`GSpShutdown` (page 40)

`GSpStartup` (page 40)

`InvokeDSpCallbackUPP` (page 41)

`InvokeDSpEventUPP` (page 41)

`ISpAllocateADBDeferBlock` (page 41)

`ISpDevice_Dispose` (page 47)

`ISpDevice_New` (page 49)

`ISpDisposeADBDeferBlock` (page 49)

`ISpDriver_CheckConfiguration` (page 50)

`ISpDriver_DisposeDevices` (page 50)

`ISpDriver_FindAndLoadDevices` (page 50)

`ISpDriver_Tickle` (page 51)

`ISpElement_Dispose` (page 58)

`ISpElement_New` (page 63)

Functions

DisposeDSpCallbackUPP

Unsupported

```
void DisposeDSpCallbackUPP (
    DSpCallbackUPP userUPP
);
```

Parameters

userUPP

Return Value

Discussion

Version Notes

Declared In

DrawSprocket.h

DisposeDSpEventUPP

Unsupported

```
void DisposeDSpEventUPP (
    DSpEventUPP userUPP
);
```

Parameters

userUPP

Return Value

Discussion

Version Notes

Declared In
DrawSprocket.h

DSpAltBuffer_Dispose

Disposes of an alternate buffer.

Unsupported

```
OSStatus DSpAltBuffer_Dispose (
    DSpAltBufferReference inAltBuffer
);
```

Parameters

inAltBuffer
A reference to the buffer to dispose.

Return Value

Discussion

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
DrawSprocket.h

DSpAltBuffer_GetCGrafPtr

Obtains the drawing area for an alternate buffer.

Unsupported

```
OSStatus DSpAltBuffer_GetCGrafPtr (
    DSpAltBufferReference inAltBuffer,
    DSpBufferKind inBufferKind,
    CGrafPtr *outCGrafPtr,
    GDHandle *outGDevice
);
```

Parameters

inAltBuffer

A reference to an alternate buffer.

inBufferKind

The kind of buffer. Currently the only supported buffer kind is `kDSpBufferKind_Normal`.

outCGrafPtr

On return, the graphics pointer associated with an alternate buffer.

outGDevice

On return, a pointer to the graphics device associated with the `CGrafPort`.

Return Value

Discussion

You must set both the proper graphics port (`CGrafPort`) and graphics device (`GDevice`) when you draw to DrawSprocket buffers to account for cases of multiple monitors and the like.

After the `DSpAltBuffer_GetCGrafPtr` function returns, you can use the pointer indicated in `outCGrafPtr` to draw into the alternate buffer. After drawing into the alternate buffer, you should invalidate the rectangles that you have worked in using the function `DSpAltBuffer_InvalRect` (page 27).

Special Considerations

Do not call at interrupt time.

Version Notes

Introduced with DrawSprocket 1.0.

Declared In

`DrawSprocket.h`

DSpAltBuffer_InvalRect

Invalidates a rectangle in an alternate buffer.

Unsupported

```
OSStatus DSpAltBuffer_InvalRect (
    DSpAltBufferReference inAltBuffer,
    const Rect *inInvalidRect
);
```

Parameters

inAltBuffer

A reference to an alternate buffer.

inInvalidRect

A pointer to the rectangle to be invalidated.

Return Value

Discussion
For example, you must invalidate areas of an underlay you have changed so that the changes are transferred to the back buffer on the next call to `DSpContextSwapBuffers`.

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

DSpAltBuffer_New

Creates an alternate buffer for an underlay.

Unsupported

```
OSStatus DSpAltBuffer_New (
    DSpContextReference inContext,
    Boolean inVRAMBuffer,
    DSpAltBufferAttributes *inAttributes,
    DSpAltBufferReference *outAltBuffer
);
```

Parameters

inContext

A reference to the context for which you want to create an alternate buffer.

inVRAMBuffer

A value of `true` requests that DrawSprocket create the buffer in VRAM if possible (it may be created in the current heap). A value of `false` means to create the buffer in the current heap.

inAttributes

A pointer to a structure specifying additional attributes of the alternate buffer. See `DSpAltBuffersAttributes` for more information. If you pass `NULL`, the alternate buffer has the same attributes as the specified context.

outAltBuffer

On return, a pointer to the alternate buffer reference.

Return Value

Discussion
If you specify additional attributes in the `inAttributes` parameter, you cannot use the alternate buffer as an underlay buffer.

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

DSpBlit_Faster

Performs the specified blitting operation (including scaling).

Unsupported

```
OSStatus DSpBlit_Faster (
    DSpBlitInfoPtr inBlitInfo,
    Boolean inAsyncFlag
);
```

Parameters

inBlitInfo

A pointer to a structure that specifies the blitting operation you want to perform. See the `DSpBlitInfo` structure for more information.

inAsyncFlag

If set to `true`, DrawSprocket attempts to perform the blitting operation asynchronously.

Return Value
A result code.

Discussion
You can use this function to copy images between any two buffers that can be represented by a `CGrafPort` reference.

If you specify asynchronous blitting, you must specify a completion function in the `inBlitInfo` structure which will be called when DrawSprocket finishes the blitting operation.

Version Notes
Introduced with DrawSprocket 1.1.

Declared In
`DrawSprocket.h`

DSpBlit_Fastest

Performs the specified blitting operation (without scaling).

Unsupported

```
OSStatus DSpBlit_Fastest (
    DSpBlitInfoPtr inBlitInfo,
    Boolean inAsyncFlag
);
```

Parameters

inBlitInfo

A pointer to a structure that specifies the blitting operation you want to perform. See the `DSpBlitInfo` structure for more information.

inAsyncFlag

If set to `true`, DrawSprocket attempts to perform the blitting operation asynchronously.

Return Value
A result code.

Discussion
You can use this function to copy images between any two buffers that can be represented by a `CGrafPort` reference. Unlike `DSpBlit_Faster` (page 29) , `DSpBlit_Fastest` forgoes checking for special drawing cases (such as clipping) when copying between buffers.

If you specify asynchronous blitting, you must specify a completion function in the `inBlitInfo` structure which will be called when DrawSprocket finishes the blitting operation.

Version Notes
Introduced with DrawSprocket 1.1.

Declared In
`DrawSprocket.h`

DSpCanUserSelectContext

Determines whether there is a meaningful choice of contexts to present to the user with the `DSpUserSelectContext` function.

Unsupported

```
OSStatus DSpCanUserSelectContext (
    DSpContextAttributesPtr inDesiredAttributes,
    Boolean *outUserCanSelectContext
);
```

Parameters

`inDesiredAttributes`
A pointer to a context attributes structure that specifies the required attributes.

`outUserCanSelectContext`
On return, the value is `true` if there are multiple contexts that meet the specified attribute requirements; `false` if there are not.

Return Value
A result code.

Discussion
This function `DSpCanUserSelectContext` allows you to check whether calling `DSpUserSelectContext` is useful so as to avoid presenting the user with a selection dialog box when there is no choice of displays.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

DSpContext_Flatten

Converts a context into a format suitable for saving to disk—for example, to save user preferences.

Unsupported

```
OSStatus DSpContext_Flatten (
    DSpContextReference inContext,
    void *outFlatContext
);
```

Parameters

inContext

A reference to the context to be flattened.

outFlatContext

A pointer to the buffer to hold the flattened context. The buffer must be large enough to hold the flattened context. You can find out the correct size by calling `DSpContext_GetFlattenedSize` (page 32). On return, the buffer holds the flattened context.

Return Value

Discussion

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

DSpContext_GetDirtyRectGridSize

Finds out the current grid size for a context's dirty rectangles.

Unsupported

```
OSStatus DSpContext_GetDirtyRectGridSize (
    DSpContextReferenceConst inContext,
    UInt32 *outCellPixelWidth,
    UInt32 *outCellPixelHeight
);
```

Parameters

inContext

A reference to a context for which you want to know the current grid cell size of the dirty rectangles.

outCellPixelWidth

On return, the width of the grid cell in pixels.

outCellPixelHeight

On return, the height of the grid cell in pixels.

Return Value
A result code.

Discussion
The height and width values may be different from the values specified in `DSpContext_SetDirtyRectGridSize` because the grid cells must be multiples of the base grid size. For example, if you request a grid cell size of 40 by 40 pixels on the current PowerPC machines, the actual cell size will be 64 by 64 because the base grid size is 32 by 32 pixels. To find out the dimensions of the base grid, you can use the `DSpContext_GetDirtyRectGridUnits` (page 32) function.

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

DSpContext_GetDirtyRectGridUnits

Finds out the size of the base dirty rectangle grid for a context.

Unsupported

```
OSStatus DSpContext_GetDirtyRectGridUnits (
    DSpContextReferenceConst inContext,
    UInt32 *outCellPixelWidth,
    UInt32 *outCellPixelHeight
);
```

Parameters

inContext
: A reference to a context for whose base dirty rectangle grid size you want to determine.

outCellPixelWidth
: On return, the width of the base grid in pixels.

outCellPixelHeight
: On return, the height of the base grid in pixels.

Return Value
A result code.

Discussion
The grid unit size is based on a number of machine characteristics such as the bus width and L1 cache size. For example, on current PowerPC-based machines, the grid unit size is 32 by 32 pixels (corresponding to the 32 bytes that make up the width of a PowerPC cache line). When you specify a grid cell size with the `DSpContext_SetDirtyRectGridSize` function, DrawSprocket rounds the requested size to a multiple of the base grid unit size. For example, if you request a grid cell size of 40 by 40 pixels, the actual cell size will be 64 by 64.

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

DSpContext_GetFlattenedSize

Determines how much memory is required to store a flattened version of a context.

Unsupported

```
OSStatus DSpContext_GetFlattenedSize (
    DSpContextReference inContext,
    UInt32 *outFlatContextSize
);
```

Parameters

inContext

A reference to the context you intend to flatten.

outFlatContextSize

On return, the number of bytes required to store a flattened version of the context.

Return Value

Discussion

After calling the `DSpContext_GetFlattenedSize` function, you can then allocate a buffer of `outFlatContextSize` size and pass it to `DSpContext_Flatten` (page 30).

Special Considerations

Do not call at interrupt time.

Version Notes

Introduced with DrawSprocket 1.0.

Declared In

`DrawSprocket.h`

DSpContext_GetMaxFrameRate

Obtains the maximum frame rate for a specified context.

Unsupported

```
OSStatus DSpContext_GetMaxFrameRate (
    DSpContextReferenceConst inContext,
    UInt32 *outMaxFPS
);
```

Parameters

inContext

A reference to the context whose maximum frame rate you want to get.

outMaxFPS

On return, the maximum frame rate in frames per second for the context specified in the `inContext` parameter. The frame rate given is not necessarily the same as the maximum frame rate passed by the most recent call to the `DSpContext_SetMaxFrameRate` function. If 0 is given as the maximum frame rate, there are no frame rate restrictions in place.

Return Value

A result code.

Discussion

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

DSpContext_GetUnderlayAltBuffer

Obtains the current underlay associated with a context.

Unsupported

```
OSStatus DSpContext_GetUnderlayAltBuffer (
    DSpContextReferenceConst inContext,
    DSpAltBufferReference *outUnderlay
);
```

Parameters

inContext
A reference to the context whose underlay you want to get.

outUnderlay
On return, a reference to the alternate buffer that holds the underlay.

Return Value

Discussion

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

DSpContext_InvalBackBufferRect

Invalidates a specific area of a context's back buffer, so that only a portion of the screen needs to be redrawn when the buffers are next swapped.

Unsupported

```
OSStatus DSpContext_InvalBackBufferRect (
    DSpContextReference inContext,
    const Rect *inRect
);
```

Parameters

inContext
A reference to the context whose back buffer is to be invalidated.

inRect
A pointer to a rectangle specifying the area (in back-buffer coordinates) to invalidate.

Return Value
A result code.

Discussion
If you do not call this function between buffer swaps, the entire back buffer is considered invalid when a swap occurs. The invalid rectangles must be set prior to each call to `DSpContext_SwapBuffers`; the dirty rectangle list is emptied before `DSpContext_GetBackBuffer` returns the back buffer for re-use.

You can make multiple calls to this function between swaps to accumulate invalid rectangular areas.

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

DSpContext_Restore

Restores a context that was saved previously, most likely to preserve a user's preferences.

Unsupported

```
OSStatus DSpContext_Restore (
    void *inFlatContext,
    DSpContextReference *outRestoredContext
);
```

Parameters

inFlatContext

A pointer to the flattened context. Typically, the context would have been saved out to disk and reloaded on a later execution of the game before calling this function.

outRestoredContext

On return, a reference to the restored context, if it exists.

Return Value

Discussion
If `DSpContext_Restore` can't find a match, the user probably has reconfigured the displays since the last time your game was run, and the call returns an error. This function has a high probability of failure, so your game should not rely on being able to restore the context. However, the game should attempt to do so as part of the normal saving of the user preferences.

If you save a context, flatten it by calling `DSpContext_Flatten` (page 30) before you first make the context's play state active otherwise, the saved data will not contain the proper information with which to locate the display.

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
DrawSprocket.h

DSpContext_SetDirtyRectGridSize

Suggests a grid size for the context's dirty rectangles.

Unsupported

```
OSStatus DSpContext_SetDirtyRectGridSize (
    DSpContextReference inContext,
    UInt32 inCellPixelWidth,
    UInt32 inCellPixelHeight
);
```

Parameters

inContext

A reference to a context whose dirty rectangle grid size you want to set.

inCellPixelWidth

The width of the grid in pixels.

inCellPixelHeight

The height of the grid in pixels.

Return Value
A result code.

Discussion
The `DSpContext_SetDirtyRectGridSize` function takes a reference to a context in the `inContext` parameter and sets the dirty rectangle grid size for that context as closely as possible to the dimensions passed in the `inCellPixelWidth` and `inCellPixelHeight` parameters. The size used depends on factors such as the L1 cache size and the CPU bus width, so your suggested values may not be the actual values used, but DrawSprocket will attempt to match your suggested size as closely as possible.

To find out what size dirty rectangle grid DrawSprocket is actually using, call `DSpContext_GetDirtyRectGridSize` (page 31). To find out the base grid size that all dirty rectangle grids must be a multiple of, use the function `DSpContext_GetDirtyRectGridUnits` (page 32).

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
DrawSprocket.h

DSpContext_SetMaxFrameRate

Sets a maximum frame rate for a specified context.

Unsupported

```
OSStatus DSpContext_SetMaxFrameRate (
    DSpContextReference inContext,
    UInt32 inMaxFPS
);
```

Parameters

inContext

A reference to the context whose maximum frame rate you want to set.

inMaxFPS

The maximum frame rate in frames per second.

Return Value
A result code.

Discussion
A call to the function `DSpContext_SetMaxFrameRate` does not guarantee that your game will achieve the maximum rate, but if it attempts to exceed the rate, DrawSprocket will slow down the buffer swapping.

The actual frame rate that is set is not necessarily the frame rate you specified, because DrawSprocket internally converts the specified maximum frame rate into a value that can be used to skip a number of frames for each frame that is drawn.

For example, if the monitor refresh rate is 66.7 Hz, and you request a frame rate of 30 fps, DrawSprocket internally skips every other frame, and your resulting frame rate is about 33.3 Hz.

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

DSpContext_SetUnderlayAltBuffer

Designates an alternate buffer to be used as the current underlay buffer for a context.

Unsupported

```
OSStatus DSpContext_SetUnderlayAltBuffer (
    DSpContextReference inContext,
    DSpAltBufferReference inNewUnderlay
);
```

Parameters

inContext

A reference to the context that uses the underlay.

inNewUnderlay

A reference to the alternate buffer that holds the underlay.

Return Value

Discussion
Underlay buffers are used to "clean" a back buffer when `DSpContext_GetBackBuffer` is called. When a back buffer is retrieved and there is an underlay buffer, the invalid areas in the back buffer are restored from the underlay buffer. This is most useful in sprite games, or in games where the background is static (or changes infrequently).

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

DSpContext_SetVBLProc

Piggybacks your own VBL task to a particular context.

Unsupported

```
OSStatus DSpContext_SetVBLProc (
    DSpContextReference inContext,
    DSpCallbackUPP inProcPtr,
    void *inRefCon
);
```

Parameters

inContext
The context the VBL task is associated with.

inProcPtr
A pointer to an application-supplied callback function. See the `DSpCallbackProc` callback function for more information about implementing this function.

inRefCon
A reference constant to be handed back by DrawSprocket when it calls the `inProcPtr` callback.

Return Value

Discussion
Because DrawSprocket needs to set up VBL tasks of its own, you can piggyback your own VBL task to a particular context easily with this function, instead of digging down through the system to find the correct slot ID and installing your own.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

DSpUserSelectContext

Presents a dialog box that allows the user to select a display.

Unsupported

```
OSStatus DSpUserSelectContext (
     inDesiredAttributes,
    DisplayIDType inDialogDisplayLocation,
    DSpEventUPP inEventProc,
    DSpContextReference *outContext
);
```

Parameters

inDesiredAttributes

A pointer to an attributes structure that specifies a minimum set of required display characteristics.

inDialogDisplayLocation

The ID of the display on which to present the selection dialog box. If this parameter is 0, DrawSprocket positions the dialog box on the main screen.

inEventProc

A pointer to an application-defined event-processing function that allows you to handle events received by the dialog box that DrawSprocket cannot process, such as update events, in your game context area. See the function `DSpEventProcPtr` (page 142) for more information about implementing this function.

outContext

On return, a reference to a context.

Return Value
A result code.

Discussion
In the selection dialog box all graphics devices appear, although the user can select only those contexts that meet or exceed the minimum characteristics given in the `inDesiredAttributes` parameter.

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with DrawSprocket 1.0.

Declared In
`DrawSprocket.h`

GSpConfigure

Unsupported

```
GSpConfigure (
   GSpEventProcPtr inEventProc,
   Point *inUpperLeft
);
```

Parameters

inEventProc

inUpperLeft

Return Value

Discussion

Version Notes

Declared In

GoggleSprocket.h

GSpShutdown

Unsupported

```
GSpShutdown (
   UInt32 inReserved
);
```

Parameters

inReserved

Return Value

Discussion

Version Notes

Declared In

GoggleSprocket.h

GSpStartup

Unsupported

```
GSpStartup (
   UInt32 inReserved
);
```

Parameters

inReserved

Return Value

Discussion

Version Notes

Declared In

GoggleSprocket.h

InvokeDSpCallbackUPP

Unsupported

```
Boolean InvokeDSpCallbackUPP (
    DSpContextReference inContext,
    void *inRefCon,
    DSpCallbackUPP userUPP
);
```

Parameters

inContext

userUPP

Return Value

Discussion

Version Notes

Declared In

DrawSprocket.h

InvokeDSpEventUPP

Unsupported

```
Boolean InvokeDSpEventUPP (
    EventRecord *inEvent,
    DSpEventUPP userUPP
);
```

Parameters

inEvent

userUPP

Return Value

Discussion

Version Notes

Declared In

DrawSprocket.h

ISpAllocateADBDeferBlock

Unsupported

```
OSErr ISpAllocateADBDeferBlock (
    ISpADBDeferRef *createBlock
);
```

Parameters

createBlock

Return Value

Discussion

Version Notes

Declared In
InputSprocket.h

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.

Unsupported

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

Parameters

inEventProcPtr

A pointer to an application-supplied function for handling update events. See `ISpEventProcPtr` (page 154) for more information about implementing this function.

Return Value

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 67) before calling this function.

Special Considerations

Do not call during interrupt time.

Version Notes
Introduced with InputSprocket 1.0.

Declared In
InputSprocket.h

ISpDevices_Activate

Activates the specified devices.

Unsupported

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

Parameters

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.

Return Value

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 43) instead.

Version Notes

Introduced with InputSprocket 1.0.

Declared In

`InputSprocket.h`

ISpDevices_ActivateClass

Activates the specified class of devices.

Unsupported

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

Parameters

inClass

The class of device you want to activate.

Return Value

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 42).

Version Notes

Introduced with InputSprocket 1.1.

Declared In

`InputSprocket.h`

ISpDevices_Deactivate

Deactivates the specified devices.

Unsupported

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

Parameters

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.

Return Value

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 44) instead.

Version Notes

Introduced with InputSprocket 1.0.

Declared In

`InputSprocket.h`

ISpDevices_DeactivateClass

Deactivates the specified class of devices.

Unsupported

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

Parameters

inClass

The class of device you want to deactivate.

Return Value

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 44).

Version Notes

Introduced with InputSprocket 1.1.

Declared In
`InputSprocket.h`

ISpDevices_Extract

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

Unsupported

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

Parameters

`inBufferCount`

The number of device references in the array pointed to by the `buffer` parameter.

`outCount`

On return, the `outCount` specifies the number of device references on the systemwide list.

`buffer`

On return, `buffer` points to the array of extracted device references.

Return Value

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.

Declared In
`InputSprocket.h`

ISpDevices_ExtractByClass

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

Unsupported

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

Parameters

inClass

The category of device to count and extract. See "Built-in Device Categories" (page 189) for built-in values for this parameter.

inBufferCount

The number of device references in the array pointed to by the `buffer` parameter.

outCount

On return, the `outCount` specifies the number of device references on the systemwide list.

buffer

On return, `buffer` points to the array of extracted device references.

Return Value

Discussion

If you want to extract classes of devices so you can activate or deactivate them, you should use the functions `ISpDevices_ActivateClass` (page 43) or `ISpDevices_DeactivateClass` (page 44) 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.

Declared In

`InputSprocket.h`

ISpDevices_ExtractByIdentifier

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

Unsupported

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

Parameters

inIdentifier

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.

Return Value

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.

Declared In

`InputSprocket.h`

ISpDevice_Dispose

Unsupported

```
OSStatus ISpDevice_Dispose (
    ISpDeviceReference inReference
);
```

Parameters

inReference

Return Value

Discussion

Version Notes

Declared In

`InputSprocket.h`

ISpDevice_GetDefinition

Obtains a device definition structure for a specified device.

Unsupported

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

Parameters

inDevice

The device whose device definition structure you want to get.

inBuflen

The length of buffer to hold the device definition.

outStruct

A pointer to a device definition structure. See `ISpDeviceDefinition` (page 161) for more information.

Return Value

Discussion

Version Notes
Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpDevice_GetElementList

Obtains an element list for a specified device.

Unsupported

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

Parameters

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.

Return Value

Discussion

Version Notes
Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpDevice_IsActive

Finds out if a device is active.

Unsupported

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

Parameters

inDevice

A reference to a device.

outIsActive

This value is `true` if the device is active, `false` if it is inactive.

Return Value

Discussion

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

Version Notes

Introduced with InputSprocket 1.0.

Declared In

`InputSprocket.h`

ISpDevice_New

Unsupported

```
OSStatus ISpDevice_New (
    const ISpDeviceDefinition *inStruct,
    ISpDriverFunctionPtr_MetaHandler metaHandler,
    UInt32 refCon,
    ISpDeviceReference *outReference
);
```

Parameters

inStruct

metaHandler

refCon

outReference

Return Value

Discussion

Version Notes

Declared In

`InputSprocket.h`

ISpDisposeADBDeferBlock

Unsupported

```
OSErr ISpDisposeADBDeferBlock (
    ISpADBDeferRef disposeBlock
);
```

Parameters

disposeBlock

Return Value

Discussion

Version Notes

Declared In

InputSprocket.h

ISpDriver_CheckConfiguration

Unsupported

```
OSStatus ISpDriver_CheckConfiguration (
    Boolean *validConfiguration
);
```

Parameters

validConfiguration

Return Value

Discussion

Version Notes

Declared In

InputSprocket.h

ISpDriver_DisposeDevices

Unsupported

```
OSStatus ISpDriver_DisposeDevices ();
```

Parameters

Return Value

Discussion

Version Notes

Declared In

InputSprocket.h

ISpDriver_FindAndLoadDevices

Unsupported

```
OSStatus ISpDriver_FindAndLoadDevices (
    Boolean *keepDriverLoaded
);
```

Parameters

keepDriverLoaded

Return Value

Discussion

Version Notes

Declared In

InputSprocket.h

ISpDriver_Tickle

Unsupported

```
void ISpDriver_Tickle ();
```

Parameters

Return Value

Discussion

Version Notes

Declared In

InputSprocket.h

ISpElementList_AddElements

Adds more elements to an element list.

Unsupported

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

Parameters

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.

Return Value

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.

Declared In
`InputSprocket.h`

ISpElementList_Dispose

Disposes of a specified element list.

Unsupported

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

Parameters

`inElementList`
A reference to the element list you want to dispose.

Return Value

Discussion

Special Considerations

Do not call at interrupt time.

Version Notes
Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpElementList_Extract

Extracts and counts elements in an element list.

Unsupported

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

Parameters

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.

Return Value

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.

Declared In

`InputSprocket.h`

ISpElementList_ExtractByKind

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

Unsupported

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

Parameters

inElementList

A reference to the element list containing the elements you want to extract.

inKind

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 return, the number of element references of the specified kind in the element list that were actually copied to the buffer.

buffer

On return, a pointer to an array of element references indicating the elements that match the specified kind.

Return Value

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.

Declared In

`InputSprocket.h`

ISpElementList_ExtractByLabel

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

Unsupported

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

Parameters

inElementList

A reference to the element list containing the elements you want to extract.

inLabel

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 return, the number of element references with the specified label in the element list that were actually copied to the buffer*.

buffer

On return, a pointer to an array of element references indicating the elements that match the specified label.

Return Value

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.

Declared In

`InputSprocket.h`

ISpElementList_Flush

Flushes the event queue associated with an element list.

Unsupported

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

Parameters

inElementList

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

Return Value

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.

Declared In
`InputSprocket.h`

ISpElementList_GetNextEvent

Obtains the most recent event from a list of elements.

Unsupported

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

Parameters

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, `ISpElementEventPtr` points to a structure of type `ISpElementEvent` (page 169). 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`.

Return Value

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 62) except that it operates on an element list.

Version Notes
Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpElementList_New

Creates a new element list.

Unsupported

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

Parameters

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 196).

Return Value

Discussion

Special Considerations

Do not call at interrupt time.

Version Notes

Introduced with InputSprocket 1.0.

Declared In

`InputSprocket.h`

ISpElementList_RemoveElements

Removes elements from an element list.

Unsupported

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

Parameters

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.

Return Value

Discussion

Special Considerations

Do not call at interrupt time.

Version Notes

Introduced with InputSprocket 1.0.

Declared In

`InputSprocket.h`

ISpElement_Dispose

Unsupported

```
OSStatus ISpElement_Dispose (
    ISpElementReference inElement
);
```

Parameters

inElement

Return Value

Discussion

Version Notes

Declared In

`InputSprocket.h`

ISpElement_DisposeVirtual

Disposes of virtual elements.

Unsupported

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

Parameters

count

The number of elements to dispose.

inElements

A pointer to the array of element references you want to dispose.

Return Value

Discussion

You must call the function `ISpStop` (page 71) before disposing of any elements that are receiving data.

Version Notes

Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpElement_Flush

Flushes the event queue associated with an element.

Unsupported

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

Parameters

inElement

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

Return Value

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.

Declared In

`InputSprocket.h`

ISpElement_GetComplexState

Obtains the current state of an element.

Unsupported

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

Parameters

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.

Return Value

Discussion
As most built-in element types return no larger than a 4-byte value, you should use the function `ISpElement_GetSimpleState` (page 63) instead.

Version Notes
Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpElement_GetConfigurationInfo

Obtains configuration information for an element.

Unsupported

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

Parameters

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.

Return Value

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 159) , `ISpDPadConfigurationInfo` (page 162) , and `ISpAxisConfigurationInfo` (page 159) and `ISpDeltaConfigurationInfo` (page 160) structures.

Version Notes
Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpElement_GetDevice

Finds out what device an element belongs to.

Unsupported

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

Parameters

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.

Return Value

Discussion

Version Notes
Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpElement_GetGroup

Finds out what group an element belongs to.

Unsupported

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

Parameters

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.

Return Value

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.

Declared In
`InputSprocket.h`

ISpElement_GetInfo

Obtains an element information structure for an element.

Unsupported

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

Parameters

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 169) for more details.

Return Value

Discussion

This function obtains information common to all elements (kind, label, and human-readable string). To get element kind–specific information use the function `ISpElement_GetConfigurationInfo` (page 60).

Version Notes

Introduced with InputSprocket 1.0.

Declared In

`InputSprocket.h`

ISpElement_GetNextEvent

Obtains event data for a single element.

Unsupported

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

Parameters

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 169). 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`.

Return Value

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 56) except that it operates on a single element.

Version Notes
Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpElement_GetSimpleState

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

Unsupported

```
OSStatus ISpElement_GetSimpleState (
    ISpElementReference inElement,
    UInt32 *state
);
```

Parameters

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 "Element Label Constants" (page 191).

Return Value

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 59).

Version Notes
Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpElement_New

Unsupported

```
OSStatus ISpElement_New (
    const ISpElementDefinitionStruct *inStruct,
    ISpElementReference *outElement
);
```

Parameters

inStruct

outElement

Return Value

Discussion

Version Notes

Declared In
InputSprocket.h

ISpElement_NewVirtual

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

Unsupported

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

Parameters

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 207).

Return Value

Discussion
In most cases, it is much easier to allocate elements using the function `ISpElement_NewVirtualFromNeeds` (page 64).

Version Notes
Introduced with InputSprocket 1.0.

Declared In
InputSprocket.h

ISpElement_NewVirtualFromNeeds

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

Unsupported

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

Parameters

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 207).

Return Value

Discussion

This function only works if you are using built-in element kinds, which are described in "Built-in Element Kinds" (page 190). For other element kinds, use the `ISpElement_NewVirtual` function to create virtual elements.

Note that you typically call `ISpInit` (page 67) after calling this function, passing much of the same information.

Version Notes

Introduced with InputSprocket 1.0.

Declared In

`InputSprocket.h`

ISpElement_PushComplexData

Unsupported

```
OSStatus ISpElement_PushComplexData (
    ISpElementReference inElement,
    UInt32 buflen,
    void *state,
    const AbsoluteTime *time
);
```

Parameters

inElement

buflen

time

Return Value

Discussion

Version Notes

Declared In

InputSprocket.h

ISpElement_PushSimpleData

Unsupported

```
OSStatus ISpElement_PushSimpleData (
    ISpElementReference inElement,
    UInt32 data,
    const AbsoluteTime *time
);
```

Parameters

inElement

data

time

Return Value

Discussion

Version Notes

Declared In

InputSprocket.h

ISpGetGlobalElementList

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

Unsupported

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

Parameters

outElementList

A reference to the global element list.

Return Value

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

Version Notes
Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpGetVersion

Returns the version of InputSprocket installed.

Unsupported

```
NumVersion ISpGetVersion ();
```

Parameters

Return Value
The version number of InputSprocket.

Discussion

Version Notes
Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpInit

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

Unsupported

```
OSStatus ISpInit (
    UInt32 count,
    ISpNeed *needs,
    ISpElementReference *inReferences,
    OSType appCreatorCode,
    OSType subCreatorCode,
    UInt32 flags,
    SInt16 setListResourceId,
    UInt32 reserved
);
```

Parameters

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`.

reserved

Reserved. Set to 0.

Return Value

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 64) and `ISpElement_NewVirtualFromNeeds` (page 64).

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 44).

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.

Declared In
`InputSprocket.h`

ISpInstallADBDefer

Unsupported

```
OSErr ISpInstallADBDefer (
    ISpADBDeferRef refBlock,
    ADBAddress reqAddress,
    ISpADBDeferCallbackProcPtr installProc,
    UInt32 clientRefCon,
    ADBServiceRoutineUPP *prevRoutine,
    Ptr *prevDataArea
);
```

Parameters

refBlock
reqAddress
installProc
clientRefCon
prevRoutine
prevDataArea

Return Value

Discussion

Version Notes

Declared In
`InputSprocket.h`

ISpPlotAppIconSuite

Unsupported

```
OSErr ISpPlotAppIconSuite (
    const Rect *theRect,
    IconAlignmentType align,
    IconTransformType transform,
    SInt16 iconSuiteResourceId
);
```

Parameters

theRect

align

transform

iconSuiteResourceId

Return Value

Discussion

Version Notes

Declared In

`InputSprocket.h`

ISpRemoveADBDefer

Unsupported

```
OSErr ISpRemoveADBDefer (
    ISpADBDeferRef refBlock
);
```

Parameters

refBlock

Return Value

Discussion

Version Notes

Declared In

`InputSprocket.h`

ISpResume

Resumes InputSprocket.

Unsupported

```
OSStatus ISpResume ();
```

Parameters

Return Value

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.

Declared In
`InputSprocket.h`

ISpShutdown

Shuts down InputSprocket and unloads all InputSprocket drivers.

Unsupported

```
OSStatus ISpShutdown ();
```

Parameters

Return Value

Discussion
Your application must call this function before it quits.

Version Notes
Introduced with InputSprocket 1.1.

Declared In
`InputSprocket.h`

ISpStartup

Starts up InputSprocket and loads all the InputSprocket drivers.

Unsupported

```
OSStatus ISpStartup ();
```

Parameters

Return Value

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

Version Notes
Introduced with InputSprocket 1.1.

Declared In
`InputSprocket.h`

ISpStop

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

Unsupported

```
OSStatus ISpStop ();
```

Parameters

Return Value

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 71).

Version Notes
Introduced with InputSprocket 1.0.

Declared In
`InputSprocket.h`

ISpSuspend

Suspends InputSprocket.

Unsupported

```
OSStatus ISpSuspend ();
```

Parameters

Return Value

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.

Declared In
`InputSprocket.h`

ISpTickle

Allows InputSprocket to give up time to other InputSprocket drivers.

Unsupported

```
OSStatus ISpTickle ();
```

Parameters

Return Value

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.

Declared In
`InputSprocket.h`

ISpTimeToMicroseconds

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

Unsupported

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

Parameters

`inTime`
A pointer to the `AbsoluteTime` value you want to convert.

`outMicroseconds`
On return, `outMicroseconds` points to the converted time value.

Return Value

Discussion

Version Notes
Introduced with InputSprocket 1.2.

Declared In
`InputSprocket.h`

ISpUptime

Obtains the time elapsed since machine startup.

Unsupported

```
AbsoluteTime ISpUptime ();
```

Parameters

Return Value
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.

Declared In
`InputSprocket.h`

NewDSpCallbackUPP

Unsupported

```
DSpCallbackUPP NewDSpCallbackUPP (
    DSpCallbackProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

Discussion

Version Notes

Declared In
`DrawSprocket.h`

NewDSpEventUPP

Unsupported

```
DSpEventUPP NewDSpEventUPP (
    DSpEventProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

Discussion

Version Notes

Declared In
`DrawSprocket.h`

NSpClearMessageHeader

Initializes the entire header structure.

Unsupported

```
void NSpClearMessageHeader (
    NSpMessageHeader *inMessage
);
```

Parameters

inMessage

A pointer to the message to be initialized.

Return Value

Discussion
You should call the `NSpClearMessageHeader` function each time before you start filling in your message structures. If you fail to initialize your message structures, you may end up with invalid data.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpConvertAddressReferenceToOTAddr

Obtains an Open Transport `OTAddress` from a NetSprocket `NSpAddressReference`.

Unsupported

```
OTAddress *NSpConvertAddressReferenceToOTAddr (
    NSpAddressReference inAddress
);
```

Parameters

`inAddress`
A valid `NSpAddressReference` returned from `NSpDoModalJoinDialog`.

Return Value
A valid `OTAddress`.

Discussion
Use `NSpConvertAddressReferenceToOTAddr` when you want to use the `NSpDoModalJoinDialog` function and you do not plan to use any other functions provided in NetSprocket, such as networking, group, or player functions.

When you no longer need the address reference, you can call `NSpReleaseAddressReference` (page 106) to release it.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpConvertOTAddrToAddressReference

Obtains a NetSprocket `NSpAddressReference` from an Open Transport `OTAddress`.

Unsupported

```
NSpAddressReference NSpConvertOTAddrToAddressReference
(
    OTAddress *inAddress
);
```

Parameters

inAddress

A valid (TCP/IP or AppleTalk) `OTAddress` returned from Open Transport.

Return Value

A valid `NSpAddressReference`.

Discussion

You should use this function when you do not wish to use the human interface functions provided by NetSprocket for standard hosting, browsing, and joining.

When you no longer need the address reference, do not call `NSpReleaseAddressReference` (page 106) to release it. You must dispose of the original `OTAddress` reference in the usual manner (such as by calling `DisposePtr`).

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpDoModalHostDialog

Presents your user with a default modal dialog box for hosting a game on the network.

Unsupported

```
Boolean NSpDoModalHostDialog (
    NSpProtocolListReference ioProtocolList,
    Str31 ioGameName,
    Str31 ioPlayerName,
    Str31 ioPassword,
    NSpEventProcPtr inEventProcPtr
);
```

Parameters

ioProtocolList

An opaque reference to a list of protocols. You can create an empty list that will be filled in with information about the protocols the user selects, but you cannot pass `NULL`. If you wish to preconfigure certain protocols, you can create protocol references for them, then add them to your protocol list before passing it to this function.

ioGameName

A Pascal string (maximum 31 characters) of the name of the game to be registered in NBP and displayed to users if you are using the `NSpGame_Join` function in their game. Pass an empty string (not `NULL`) if you don't want to display a default game name. The value of `ioGameName` is often obtained from a preferences setting. This field contains changes (if any) the user made to the `ioGameName` field.

ioPlayerName

A Pascal (maximum 31 characters) string of the user name (generally from a preferences setting). Pass an empty string (not `NULL`) if you do not want a default name displayed in the dialog box. This field contains any changes the user has made to the name.

ioPassword

A Pascal (maximum 31 characters) string of the password (generally from a preferences setting). Pass an empty string (not `NULL`) if you do not want a default password displayed in the dialog box. This field contains any changes the user made to the password.

inEventProcPtr

A pointer to `DialogProcUPP`, the dialog filter function for handling Mac OS events that may affect other windows you have displayed on the screen concurrently. Pass `NULL` if you do not need to receive Mac OS events while the dialog box is being displayed.

Return Value
A value of `true` if the user selected OK, `false` if the user selected Cancel.

Discussion
This function fills in the protocol list with the protocol(s) the user has selected and configures the protocol references in the list with the proper information. If the user did not cancel the dialog box, you should then pass the protocol list to the `NSpGame_Host` function.

The NetSprocket human interface functions are forward-compatible with new protocols as they become available. This means that you don't have to change your code to accommodate new protocols when joining or hosting a game.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpDoModalJoinDialog

Presents to the user a default dialog box for finding and joining a game advertised on the network.

Unsupported

```
NSpAddressReference NSpDoModalJoinDialog (
    ConstStr31Param inGameType,
    ConstStr255Param inEntityListLabel,
    Str31 ioName,
    Str31 ioPassword,
    NSpEventProcPtr inEventProcPtr
);
```

Parameters

inGameType

A Pascal (maximum 31 characters) string used to register your game's NBP (Name Binding Protocol) type. This must be the same as the one used to host a game. If you pass `NULL` or an empty string, then NetSprocket uses the game ID (as passed to `NSpInitialize` (page 88)) to search for games on the AppleTalk network.

inEntityListLabel

A Pascal string that will be displayed above the entity list in the AppleTalk panel of the dialog box, as a label for the list of available games.

ioName

A Pascal (maximum 31 characters) string of the user name (generally from a preferences setting). Pass an empty string (not `NULL`) if you do not want a default name displayed in the dialog box. This string pointed to by `ioName` will contain any changes the user made to the name on return.

ioPassword

A Pascal (maximum 31 characters) string of the password (generally from a preferences setting). Pass an empty string (not `NULL`) if you do not want a default password displayed in the dialog box. This field will contain the any changes the user made to the password on return.

inEventProcPtr

A pointer to `DialogProcUPP`, the dialog filter function for handling Mac OS events that may affect other windows you have displayed on the screen concurrently. Pass `NULL` if you do not need to receive Mac OS events while the NetSprocket dialog box is being displayed.

Return Value
An opaque reference to the protocol address selected by the user.

Discussion
If the user cancels the dialog box, the function will return `NULL`. If the user selects OK, it will return a reference to the protocol address of a game host. You should then pass this reference to the function `NSpGame_Join` (page 82). Once you have called `NSpGame_Join`, call `NSpReleaseAddressReference` (page 106) to release the reference.

The NetSprocket human interface functions are forward-compatible with new protocols as they become available. This means that you don't have to change your code to accommodate new protocols when joining or hosting a game.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpGame_Dispose

Removes a player or host from the game.

Unsupported

```
OSStatus NSpGame_Dispose (
    NSpGameReference inGame,
    NSpFlags inFlags
);
```

Parameters

inGame

An opaque reference to your game object.

inFlags

Options for leaving the game. See "Options for Hosting, Joining, and Disposing Games" (page 205) for a list of possible values.

Return Value
A result code.

Discussion
If your application is hosting the game and you pass `kNSpGameFlag_ForceTerminateGame` in the `inFlags` parameter, the game will be stopped for all participants and the game object will be deleted. However, if you do not pass `kNSpGameFlag_ForceTerminateGame`, NetSprocket will attempt to negotiate with another player to become the host. If the negotiation is successful, the other players will be notified that the host has changed and you will be dropped from the game. If the negotiation fails, `NSpGame_Dispose` returns an error and no further action is taken.

If your application is operating as a player (created by `NSpGame_Join`), the other players are notified that you are leaving the game. The game is not terminated if you make this call as a player.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpGame_EnableAdvertising

Enables or disables advertising of the game on the network.

Unsupported

```
OSStatus NSpGame_EnableAdvertising (
    NSpGameReference inGame,
    NSpProtocolReference inProtocol,
    Boolean inEnable
);
```

Parameters

inGame
: An opaque reference to your game object.

inProtocol
: An opaque reference to the protocol for which you wish to start or stop advertising. Pass `NULL` to stop advertising on all protocols.

inEnable
: A value of `true` to start advertising or `false` to stop advertising.

Return Value
A result code.

Discussion
The function `NSpGame_Host` (page 80) automatically advertises the game, unless you passed `kNSpGameFlag_DontAdvertise` in its `inFlags` field.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpGame_GetInfo

Obtains information about an available game.

Unsupported

```
OSStatus NSpGame_GetInfo (
    NSpGameReference inGame,
    NSpGameInfo *ioInfo
);
```

Parameters

inGame

A reference to the game you want to obtain information about.

ioInfo

On return, a pointer to game information. See `NSpGameInfo` (page 177) for the format of the returned information.

Return Value
A result code.

Discussion
If you are running a server capable of hosting multiple games, then you could use this function to display information about each available game. Similarly, you could use this function on a player's computer to obtain and display available games to join.

Version Notes
Introduced with NetSprocket 1.7.

Declared In
`NetSprocket.h`

NSpGame_Host

Creates a new game object that other players can then join.

Unsupported

```
OSStatus NSpGame_Host (
    NSpGameReference *outGame,
    NSpProtocolListReference inProtocolList,
    UInt32 inMaxPlayers,
    ConstStr31Param inGameName,
    ConstStr31Param inPassword,
    ConstStr31Param inPlayerName,
    NSpPlayerType inPlayerType,
    NSpTopology inTopology,
    NSpFlags inFlags
);
```

Parameters

outGame

The address of a game reference which will be filled in by this function. Upon successful return, it will contain a pointer to the newly created game object. This field is invalid if the function returns anything other than `noErr`.

inProtocolList

An opaque reference to a list of protocols that has been returned from `DoModalHostDialog`, or created by you in your own application for advertising your game on the network.

inMaxPlayers

The maximum number of players permitted to join the game. If you want to allow unlimited players, set this value to 0. NetSprocket is more efficient when the maximum number of players is set in the `inMaxPlayers` field. The number of allowed groups does not affect the maximum number of players.

inGameName

A Pascal string containing the name of the game that will appear in game browsers. You must pass a valid Pascal string in this field.

inPassword

The password that prospective players must match to join the game. Players who do not enter a correct password will not be allowed to join. Pass `NULL` if you do not require a password for players joining your game.

inPlayerName

The name of the player hosting the game. If there is no player associated with the computer hosting the game (for example, if the computer is a dedicated game server), you should pass `NULL`.

inPlayerType

The player type, which is used only if there is a player associated with the application hosting the game. This parameter is stored in NetSprocket's player information table and may be used by the game application. It is not used by NetSprocket.

inTopology

A constant indicating the topology to use in the game. The only topology implemented in version 1.0 is client/server, indicated by the constant `kNSpClientServer`.

inFlags

Options for creating the new game object. The only currently permissible value of `inFlags` in NetSprocket is `kNSpGameFlag_DontAdvertise`, which causes the `NSpGame_Host` function to create a game object, but not actually advertise the game on the network.

Return Value

Discussion

You use this function when your application hosts a game.

Once the game is created, the game will automatically be advertised over the protocols in the protocol list.

When you have created a game object by calling `NSpGame_Host`, you will pass the game object to other host functions you call. Do not use this function for joining games; you should use the `NSpGame_Join` (page 82) function instead.

`NSpGame_Host` will return `noErr` upon successful completion, placing the new game object in the `outGame` parameter. If the game could not be created for some reason, the `NSpGameReference` will be invalid (`NULL`). You should check the result code and determine the appropriate course of action.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpGame_Join

Joins a game specified by an address.

Unsupported

```
OSStatus NSpGame_Join (
    NSpGameReference *outGame,
    NSpAddressReference inAddress,
    ConstStr31Param inName,
    ConstStr31Param inPassword,
    NSpPlayerType inType,
    void *inCustomData,
    UInt32 inCustomDataLen,
    NSpFlags inFlags
);
```

Parameters

outGame

A pointer to a game reference structure that is filled in by the function. You must provide a pointer to an `NSpGameReference` in the `outGame` parameter. This pointer will be filled in with a valid `NSpGameReference` on return.

inAddress

A valid address reference returned from the `NSpDoModalJoinDialog` function or created by the application.

inName

The player's name as it will appear to other players in the game. You must pass a valid Pascal string. `NULL` is not permitted in this field.

inPassword

The password entered by the user to join the game. Pass `NULL` or an empty string if no password is required.

inType

The player's type. This value is for your own use in classifying players. It is stored, but not used by NetSprocket.

inCustomData

The length of custom authentication data being sent to the host as part of the join request. If your game does not use a custom authentication mechanism, you must set the value to 0.

inCustomDataLen

A pointer to custom data being sent to the host for use by your custom authentication function. This parameter is passed to the host, but not used by NetSprocket. If your game does not use a custom authentication mechanism, you should set this value to `NULL`.

inFlags

Options for joining the game. There are no options for this field as of NetSprocket version 1.7.

Return Value

Discussion

This function joins the game specified by the `inAddress` parameter. You can obtain an address reference from `NSpDoModalJoinDialog` (page 77) or `NSpConvertOTAddrToAddressReference` (page 75).

Version Notes

Introduced with NetSprocket 1.0.

Declared In
NetSprocket.h

NSpGetCurrentTimeStamp

Compares time stamps.

Unsupported

```
UInt32 NSpGetCurrentTimeStamp (
    NSpGameReference inGame
);
```

Parameters

inGame

An opaque reference to your game object.

Return Value
The time value in milliseconds.

Discussion
You can use this function to compare the time stamp of a message with the current time stamp to determine how long ago a message was sent. This value is only as accurate as the round-trip time to the application hosting the game. This is a normalized value established by the server. That is, anyone in the current game who calls this function will get the same value.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
NetSprocket.h

NSpGetVersion

Returns the version of NetSprocket.

Unsupported

```
NumVersion NSpGetVersion ();
```

Parameters

Return Value
The version of NetSprocket.

Discussion

Version Notes
Introduced with NetSprocket 1.0.3.

Declared In
NetSprocket.h

NSpGroup_AddPlayer

Adds a player to a group.

Unsupported

```
OSStatus NSpGroup_AddPlayer (
    NSpGameReference inGame,
    NSpGroupID inGroupID,
    NSpPlayerID inPlayerID
);
```

Parameters

inGame

An opaque reference to your game object.

inGroupID

The group to which you are adding the player.

inPlayerID

The player to be added.

Return Value

Discussion

Version Notes
Introduced with NetSprocket 1.0.

Declared In
NetSprocket.h

NSpGroup_Dispose

Removes a group from the game.

Unsupported

```
OSStatus NSpGroup_Dispose (
    NSpGameReference inGame,
    NSpGroupID inGroupID
);
```

Parameters

inGame

An opaque reference to your game object.

inGroupID

The ID of the group to delete.

Return Value

Discussion
`NSpGroup_Dispose` does not delete the players in the group. It simply deletes the group ID. A deleted group is no longer usable by any player in the game.

NetSprocket returns an error if it could not delete the group.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
NetSprocket.h

NSpGroup_GetEnumeration

Obtains a list of the groups in the game.

Unsupported

```
OSStatus NSpGroup_GetEnumeration (
    NSpGameReference inGame,
    NSpGroupEnumerationPtr *outGroups
);
```

Parameters

inGame

An opaque reference to your game object.

outGroups

A pointer to the group enumeration structure that is allocated and populated by NetSprocket.

Return Value

Discussion
For efficient memory management, the group enumeration structure should be released by NetSprocket by calling `NSpGroup_ReleaseEnumeration` (page 86).

NetSprocket returns an error if it could not build the group list.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
NetSprocket.h

NSpGroup_GetInfo

Obtains the group's information structure.

Unsupported

```
OSStatus NSpGroup_GetInfo (
    NSpGameReference inGame,
    NSpGroupID inGroupID,
    NSpGroupInfoPtr *outInfo
);
```

Parameters

inGame

An opaque reference to your game object.

inGroupID

The group you want information about.

outInfo

A pointer to an array of group information structures.

Return Value

Discussion
The group information data structure will be allocated by NetSprocket and the structure will be populated with the group's information. When you have finished with the `NSpGroupInfo` data structure, you should release it by calling `NSpGroup_ReleaseInfo` (page 87).

NetSprocket returns an error if NetSprocket could not build the group information data structure or if the group ID was invalid.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpGroup_New

Creates a new group of players.

Unsupported

```
OSStatus NSpGroup_New (
    NSpGameReference inGame,
    NSpGroupID *outGroupID
);
```

Parameters

inGame
An opaque reference to your game object.

outGroupID
A unique number identifying the new group you have created.

Return Value

Discussion
Once a group is created, the value in the `outGroupID` parameter is distributed to each player in the game. This group ID value is independent of the network transport used. Any player in the game can use the `outGroupID` parameter to send messages to the players in the group.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpGroup_ReleaseEnumeration

Releases memory held by the group enumeration structure.

Unsupported

```
void NSpGroup_ReleaseEnumeration (
    NSpGameReference inGame,
    NSpGroupEnumerationPtr inGroups
);
```

Parameters

inGame

An opaque reference to your game object.

inGroups

A pointer to a group enumeration structure.

Return Value

Discussion

For each `NSpPlayer_GetEnumeration` (page 95) call, you should execute a corresponding `NSpGroup_ReleaseEnumeration` call to release the memory held by the structure.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpGroup_ReleaseInfo

Releases memory held by the group information structure.

Unsupported

```
void NSpGroup_ReleaseInfo (
    NSpGameReference inGame,
    NSpGroupInfoPtr inInfo
);
```

Parameters

inGame

An opaque reference to your game object.

inInfo

A pointer to an array of group information structures.

Return Value

Discussion

For each `NSpGroup_GetInfo` call, you should execute a corresponding `NSpGroup_ReleaseInfo` call to release the memory held by the group information structure.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpGroup_RemovePlayer

Removes a player from a group.

Unsupported

```
OSStatus NSpGroup_RemovePlayer (
    NSpGameReference inGame,
    NSpGroupID inGroupID,
    NSpPlayerID inPlayerID
);
```

Parameters

inGame

An opaque reference to your game object.

inGroupID

The group from which the player is to be removed.

inPlayerID

The player to be removed.

Return Value

Discussion

NetSprocket returns an error if the `NSpGroup_RemovePlayer` function could not remove the player or if the player ID or group ID is invalid. This function does not remove the player from the game. It only removes the player from the list of players in the group.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpInitialize

Initializes the NetSprocket library.

Unsupported

```
OSStatus NSpInitialize (
    UInt32 inStandardMessageSize,
    UInt32 inBufferSize,
    UInt32 inQElements,
    NSpGameID inGameID,
    UInt32 inTimeout
);
```

Parameters

inStandardMessageSize

This value is the maximum size (in bytes) of each message you expect to send regularly. For example, if your game is sending keyboard state most of the time and the keyboard state message is 40 bytes long (including the `NSpMessageHeader`) then you should set this value to 40. NetSprocket uses this value to optimize the message receipt buffers. Typically, games send messages that are either relatively constant in size, or the size of the message is proportional to the number of players. If your game doesn't have a typical message size, or if you want NetSprocket to choose a size for you, set this parameter to 0. Setting this value greater than 586 bytes while using AppleTalk may force NetSprocket to use multiple packets for sending the message and potentially decrease game performance.

inBufferSize

The number of bytes that NetSprocket will allocate in its interrupt-safe memory pool for networking during initialization. Usually, 200 KB or more is recommended for most games. You can approximate the networking pool with this formula: ((size of standard message * (send frequency or get frequency)) * max players) + 50 KB safety padding). NetSprocket cannot allocate memory at interrupt time. If you do not plan to call NetSprocket functions at interrupt time or use the asynchronous functions, or if you want NetSprocket to allocate the default amount (currently 400 KB), set this value to 0. Because NetSprocket is unable to grow its buffer after initialization, it is important to allocate enough memory in NetSprocket to send, receive, and queue the messages your game will be using.

inQElements

The maximum number of queue elements that NetSprocket will allocate. The queue elements are used to store messages until you receive them from NetSprocket; the more frequently you check for messages, the fewer queue elements you need to allocate. NetSprocket can automatically expand its message queue, if necessary, but this will degrade performance. Specifying a small number (less than 10) will use less memory, but may cause messages to be discarded due to lack of buffer space. Specifying a larger number (greater than 20) will allow you to call `NSpMessage_Get` less often and more efficiently.

inGameID

A unique identifier for your game, typically your application's creator ID. For instance, if you do not specify an NBP (Name Binding Protocol) type to NetSprocket when registering a game on an AppleTalk network, it will use this ID instead.

inTimeout

Currently unused. Pass 0 for this parameter.

Return Value

Discussion

You must initialize NetSprocket before you can call functions from the NetSprocket library.

This function may fail under a variety of circumstances, including the failure to allocate enough application memory, insufficient system memory, or failure to initialize networking in the Mac OS.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpInstallAsyncMessageHandler

Installs a message handler for your game object.

Unsupported

```
OSStatus NSpInstallAsyncMessageHandler (
    NSpMessageHandlerProcPtr inHandler,
    void *inContext
);
```

Parameters

inHandler

A pointer to your message handling function. See `NSpMessageHandlerProcPtr` (page 156) for more information about implementing this function.

inContext
The pointer that NetSprocket will pass to your message handling function.

Return Value

Discussion
You do not need to install a message handler, unless you want NetSprocket to call your handler function back as soon as a completed message has arrived. The message handler is called whenever NetSprocket receives an incoming message.

Your message handler should be in place and ready to receive messages before this function returns. NetSprocket returns an error if there was a problem installing the handler.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpInstallCallbackHandler

Unsupported

```
OSStatus NSpInstallCallbackHandler (
    NSpCallbackProcPtr inHandler,
    void *inContext
);
```

Parameters
inHandler

Return Value

Discussion

Version Notes

Declared In
`NetSprocket.h`

NSpInstallJoinRequestHandler

Installs the application-defined join request handler.

Unsupported

```
OSStatus NSpInstallJoinRequestHandler (
    NSpJoinRequestHandlerProcPtr inHandler,
    void *inContext
);
```

Parameters
inHandler
A pointer to your join request function.

inContext
A pointer that will be passed to your handler when it is called by NetSprocket.

Return Value

Discussion
You can use the `NSpInstallJoinRequestHandler` function to install a special function to process join requests for your game object. When your custom function is installed, NetSprocket will call this function whenever a join request occurs. You do not need to develop and install custom join request handlers if the NetSprocket functions already meet your requirements.

You can install a custom join request handler to override the standard authentication method of NetSprocket. By default, when a NetSprocket host receives a join request, it will first make sure that the maximum number of players has not been exceeded. Then, it will check the prospective player's password (if required) and admit the player if the password matches.

When you override this behavior, your join request function is called and passed the `NSpJoinRequestMessage` (page 180) sent by the player who wants to join. You must decide whether or not to allow the player to join, based on whatever criteria you desire. Your function must return a Boolean value to indicate whether the player can join the game.

After your custom join request handler has been installed, any subsequent join requests will be passed to this function for processing.

Also note that since the maximum round-trip time is specified when hosting a game, requests from prospective players who do not meet the maximum criterion will not be passed to your game.

NetSprocket returns an error if there was a problem installing the handler.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpMessage_Get

Receives messages that have been delivered to your game.

Unsupported

```
NSpMessageHeader *NSpMessage_Get (
    NSpGameReference inGame
);
```

Parameters

inGame

An opaque reference to your game object.

Return Value
A pointer to your incoming message data structure.

Discussion
You can use this function to retrieve and process messages whether you are a player in the game or you are hosting a game.

Once game play has begun, you will probably want to call this function each time you pass through your game loop to process all network messages as quickly and efficiently as possible.

`NSpMessage_Get` returns `NULL` if there are no messages pending. If a message has been received, NetSprocket will return a pointer to a message structure.

`NSpMessage_Get` returns a pointer to an `NSpMessageHeader` -based structure that is allocated by NetSprocket. You should call `NSpMessage_Release` to release the memory back to NetSprocket when you're done with the message. Failure to release memory in a timely fashion will limit NetSprocket's ability to handle more incoming messages. `NSpMessage_Get` and `NSpMessage_Release` are a more efficient method of message processing than the time-consuming process of copying incoming messages from NetSprocket into your application's message buffer.

You should call `NSpMessage_Get` as frequently as you can to get messages that have been sent to your player.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpMessage_Release

Releases a message obtained by calling `NSpMessage_Get`.

Unsupported

```
void NSpMessage_Release (
    NSpGameReference inGame,
    NSpMessageHeader *inMessage
);
```

Parameters

inGame
: An opaque reference to your game object.

inMessage
: A pointer to the message to be released.

Return Value

Discussion
When you have finished processing a message, you should call `NSpMessage_Release` to release the memory allocated for it.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpMessage_Send

Delivers a message to other players in the game.

Unsupported

```
OSStatus NSpMessage_Send (
    NSpGameReference inGame,
    NSpMessageHeader *inMessage,
    NSpFlags inFlags
);
```

Parameters

inGame

An opaque reference to your game object.

inMessage

A pointer to the message you want to deliver. This structure can contain any data your game requires, provided that it begins with a `NSpMessageHeader`. The header must contain valid information about the intended recipient and the size of the message. To impose a reasonable amount of type-safety, you must pass `&myStruct.headerField` to ensure the structure contains an `NSpMessageHeader` as its first element.

inFlags

Flags that specify how the message should be sent, as specified in the message header structure. See "Network Message Priority Flags" (page 203) and "Network Message Delivery Flags" (page 202) for more information.

Return Value

Discussion

Before calling this function, you must fill out the message header and message. To send a message and have the message header created for you, call the function `NSpMessage_SendTo` (page 93) instead.

Although there is no restriction on the size of your message, extremely large messages (about 50 percent of the memory allocated to NetSprocket at initialization) may not be delivered if the receiver lacks the memory to process your message.

NetSprocket will return an error if it was unable to deliver your message.

Note that `NSpMessage_Send` may return `noErr`, even though the intended recipient did not receive the message. Depending on the options you have chosen and other network conditions beyond the knowledge or control of the application, the message may not have been received by its intended recipients.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpMessage_SendTo

Creates a message header and sends a message to other players in the game.

Unsupported

```
OSStatus NSpMessage_SendTo (
    NSpGameReference inGame,
    NSpPlayerID inTo,
    SInt32 inWhat,
    void *inData,
    UInt32 inDataLen,
    NSpFlags inFlags
);
```

Parameters

inGame

An opaque reference to your game object.

inTo

The ID of the player to whom you want to send the message.

inWhat

An integer indicating the type of message to be sent. See "Network Message Type Constants" (page 204) for a listing of possible types.

inData

A pointer to the message to send.

inDataLen

The length of the message in bytes.

inFlags

Flags that specify how the message should be sent. See "Network Message Priority Flags" (page 203) and "Network Message Delivery Flags" (page 202) for more information.

Return Value

Discussion

Unlike the `NSpMessage_Send` (page 92) function, `NSpMessage_SendTo` creates a message header based on the information you pass to it. Otherwise it functions identically to `NSpMessage_Send` (page 92).

Version Notes

Introduced with NetSprocket 1.7.

Declared In

`NetSprocket.h`

NSpPlayer_ChangeType

Changes the player's type.

Unsupported

```
OSStatus NSpPlayer_ChangeType (
    NSpGameReference inGame,
    NSpPlayerID inPlayerID,
    NSpPlayerType inNewType
);
```

Parameters

inGame

An opaque reference to your game object.

inPlayerID
The ID of the player whose player type you want to change.

inNewType
The new type to assign. The player type is an arbitrary integer that you can use to help classify players. For example, in a particular game, you may assign a type to indicate players who are wounded or immobilized.

Return Value

Discussion

Version Notes
Introduced with NetSprocket 1.7.

Declared In
`NetSprocket.h`

NSpPlayer_GetAddress

Obtains a player's network address.

Unsupported

```
OSStatus NSpPlayer_GetAddress (
    NSpGameReference inGame,
    NSpPlayerID inPlayerID,
    OTAddress **outAddress
);
```

Parameters

inGame
An opaque reference to your game object.

inPlayerID
The ID of the player whose network address you want to determine.

outAddress
On return, a pointer to the TCP/IP or Appletalk `OTAddress` of the player, as returned by Open Transport.

Return Value

Discussion
You can call the function `NSpConvertOTAddrToAddressReference` (page 75) to convert the returned `OTAddress` to an address of type `NSpAddressReference`. Note however, that to release the memory associated with the address, you must call `DisposePtr`, not `NSpReleaseAddressReference` (page 106).

Version Notes
Introduced with NetSprocket 1.7.

Declared In
`NetSprocket.h`

NSpPlayer_GetEnumeration

Takes a snapshot that describes each player currently in the game.

Unsupported

```
OSStatus NSpPlayer_GetEnumeration (
    NSpGameReference inGame,
    NSpPlayerEnumerationPtr *outPlayers
);
```

Parameters

inGame

An opaque reference to your game object.

outPlayers

A pointer to a player enumeration structure which is allocated and set by NetSprocket.

Return Value

Discussion

`NSpPlayer_GetEnumeration` places the information on each player in the player enumeration structure. This structure is made available to your game via `NSpPlayerEnumerationPtr`.

It is important to release the memory held by the player enumeration structure by calling the `NSpPlayer_ReleaseEnumeration` function when you are done.

If there was a problem getting the player information, NetSprocket returns an error; in such cases the value of `outPlayers` is invalid.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpPlayer_GetInfo

Obtains information about a player.

Unsupported

```
OSStatus NSpPlayer_GetInfo (
    NSpGameReference inGame,
    NSpPlayerID inPlayerID,
    NSpPlayerInfoPtr *outInfo
);
```

Parameters

inGame

An opaque reference to your game object.

inPlayerID

The ID of the player you want information about.

outInfo

A pointer to `NSpPlayerInfoPtr` which contains a pointer to the player's information data structure you have requested.

Return Value

Discussion

When you are done with the player's information, you should call `NSpPlayer_ReleaseInfo` (page 99) to release memory associated with the structure.

NetSprocket returns an error if it could not obtain the player's information.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpPlayer_GetMyID

Obtains the ID of the player associated with the game object on the current computer.

Unsupported

```
NSpPlayerID NSpPlayer_GetMyID (
    NSpGameReference inGame
);
```

Parameters

inGame

An opaque reference to your game object.

Return Value
A valid player ID. NetSprocket returns 0 if there is no player associated with the game object.

Discussion

Version Notes

Declared In
`NetSprocket.h`

NSpPlayer_GetRoundTripTime

Unsupported

```
UInt32 NSpPlayer_GetRoundTripTime (
    NSpGameReference inGame,
    NSpPlayerID inPlayer
);
```

Parameters

inGame

inPlayer

Return Value

Discussion

Version Notes

Declared In
`NetSprocket.h`

NSpPlayer_GetThruput

Determines the data throughput between the caller and the specified player.

Unsupported

```
UInt32 NSpPlayer_GetThruput (
    NSpGameReference inGame,
    NSpPlayerID inPlayer
);
```

Parameters

inGame

An opaque reference to your game object.

inPlayer

The ID of the player you are sending the test message to.

Return Value

The throughput between the caller and the player. Throughput is measured in bytes per second.

Discussion

This function is synchronous. That is, it blocks until it finishes testing throughput unless the timeout is reached. If time-out is exceeded, -1 will be returned. Throughput between any two players may vary greatly during the course of a game.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpPlayer_ReleaseEnumeration

Releases the player enumeration structure.

Unsupported

```
void NSpPlayer_ReleaseEnumeration (
    NSpGameReference inGame,
    NSpPlayerEnumerationPtr inPlayers
);
```

Parameters

inGame

An opaque reference to your game object.

inPlayers

The player enumeration structure obtained from `NSpPlayer_GetEnumeration`.

Return Value

Discussion

For each `NSpPlayer_GetEnumeration` (page 95) call, you should execute a corresponding `NSpPlayer_ReleaseEnumeration` call to release the player enumeration structure when you no longer need it.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpPlayer_ReleaseInfo

Releases a player information structure obtained by the `NSpPlayer_GetInfo` function.

Unsupported

```
void NSpPlayer_ReleaseInfo (
    NSpGameReference inGame,
    NSpPlayerInfoPtr inInfo
);
```

Parameters

inGame

An opaque reference to your game object.

inInfo

The information structure you want to release.

Return Value

Discussion

You should use the `NSpPlayer_ReleaseInfo` function to release each player information structure obtained by `NSpPlayer_GetInfo` (page 96).

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpPlayer_Remove

Removes a player.

Unsupported

```
OSStatus NSpPlayer_Remove (
    NSpGameReference inGame,
    NSpPlayerID inPlayerID
);
```

Parameters

inGame

An opaque reference to your game object.

inPlayerID

The ID of the player you want to remove.

Return Value

Discussion

Unlike the function `NSpGame_Dispose` (page 78) , `NSpPlayer_Remove` forcibly removes a player from the game. You can call this function only when the application is hosting the game.

Version Notes

Introduced with NetSprocket 1.7.

Declared In

`NetSprocket.h`

NSpProtocolList_Append

Adds a new protocol reference to the list.

Unsupported

```
OSStatus NSpProtocolList_Append (
    NSpProtocolListReference inProtocolList,
    NSpProtocolReference inProtocolRef
);
```

Parameters

inProtocolList

An opaque reference to a protocol list.

inProtocolRef

An opaque reference to the protocol being appended.

Return Value

Discussion

The specified protocol reference is appended to the list of protocol references. Note that after appending, the reference becomes the property of the list; you cannot call `NSpProtocol_Dispose` (page 105) to delete a protocol reference in the list.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpProtocolList_Dispose

Deletes a protocol list.

Unsupported

```
void NSpProtocolList_Dispose (
    NSpProtocolListReference inProtocolList
);
```

Parameters

inProtocolList

An opaque reference to a list of protocols. When you use `NSpProtocolList_Dispose` to delete a protocol list, all the protocol references in it are deleted.

Return Value

Discussion

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpProtocolList_GetCount

Returns the number of protocol references in the list.

Unsupported

```
UInt32 NSpProtocolList_GetCount (
    NSpProtocolListReference inProtocolList
);
```

Parameters

inProtocolList

An opaque reference to a protocol list.

Return Value

The number of protocol references in the list.

Discussion

Use this function when iterating through the protocol list.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpProtocolList_GetIndexedRef

Receives the protocol reference at the indicated location in the list.

Unsupported

```
NSpProtocolReference NSpProtocolList_GetIndexedRef
(
    NSpProtocolListReference inProtocolList,
    UInt32 inIndex
);
```

Parameters

inProtocolList

An opaque reference to a list of protocols.

inIndex

A valid index entry. The index is zero-based.

Return Value

The protocol reference at the specified index.

Discussion

`NSpProtocolList_GetIndexedRef` does not remove the protocol from the list, so you must not delete its reference.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpProtocolList_New

Creates a new list for storing multiple protocol references.

Unsupported

```
OSStatus NSpProtocolList_New (
    NSpProtocolReference inProtocolRef,
    NSpProtocolListReference *outList
);
```

Parameters

inProtocolRef

An opaque reference to the protocol reference to be added to the list when it is created. Pass `NULL` if you don't want to add any protocol references at this time.

outList

An opaque reference to the protocol list that was created. This is only valid if the function returns `noErr`.

Return Value

Discussion

The `NSpGame_Host` function requires a list of protocol references, so that the game can be hosted on multiple protocols. Also, the `NSpDoModalHostDialog` function requires you to pass a protocol list that it fills in. Once a protocol reference has been added to a list, its memory belongs to the list.

Version Notes

Introduced with NetSprocket 1.0.

Declared In

`NetSprocket.h`

NSpProtocolList_Remove

Removes a protocol reference from the list.

Unsupported

```
OSStatus NSpProtocolList_Remove (
    NSpProtocolListReference inProtocolList,
    NSpProtocolReference inProtocolRef
);
```

Parameters

inProtocolList

An opaque reference to a protocol list.

inProtocolRef

An opaque reference to the protocol you are removing.

Return Value

Discussion

When a protocol reference is removed from a protocol list, its memory once again belongs to the application and should be released with a call to `NSpProtocol_Dispose` (page 105).

Version Notes

Introduced with NetSprocket 1.0.

Declared In
NetSprocket.h

NSpProtocolList_RemoveIndexed

Removes the protocol reference at a specific location in the list.

Unsupported

```
OSStatus NSpProtocolList_RemoveIndexed (
    NSpProtocolListReference inProtocolList,
    UInt32 inIndex
);
```

Parameters

inProtocolList
An opaque reference to a protocol list.

inIndex
The index entry to be removed. The index is zero-based.

Return Value

Discussion
This function is usually used in conjunction with the `NSpProtocolList_GetCount` (page 101) function for stepping through a protocol list and removing a specific protocol reference.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
NetSprocket.h

NSpProtocol_CreateAppleTalk

Creates an AppleTalk protocol reference using the specified parameters.

Unsupported

```
NSpProtocolReference NSpProtocol_CreateAppleTalk (
    ConstStr31Param inNBPName,
    ConstStr31Param inNBPType,
    UInt32 inMaxRTT,
    UInt32 inMinThruput
);
```

Parameters

inNBPName
The Name Binding Protocol name you wish users to see when browsing the AppleTalk network.

inNBPType
The Name Binding Protocol type to use when advertising the game on an AppleTalk network. This name should be representative of your game, but is never displayed to users. This name must be the same as the one you use in the `ioGameType` field of the `NSpGame_Join` function.

inMaxRTT

The maximum round-trip time (RTT) allowed for new players. Pass 0 if you do not wish to have round-trip time checked. This does not guarantee that RTT will remain at the level it is when the player joins. RTT is in milliseconds.

inMinThruput

The minimum throughput required of any prospective entrant into the game. Pass 0 if you do not wish to have throughput checked. This does not guarantee that throughput will remain at the level it is when the player joins. Throughput is measured in bytes per second.

Return Value
A reference to the created protocol, or `NULL` if there was an error in specifying the protocol.

Discussion
Use this function if you wish to preconfigure the AppleTalk protocol before calling `NSpDoModalHostDialog`, or if you want to host the game programmatically.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpProtocol_CreateIP

Creates an IP protocol reference.

Unsupported

```
NSpProtocolReference NSpProtocol_CreateIP (
    InetPort inPort,
    UInt32 inMaxRTT,
    UInt32 inMinThruput
);
```

Parameters

inPort

The port on which you wish to listen for new players. Since there is no dynamic name lookup in IP, prospective players cannot know what port a game is being played on unless they receive that information from the hosting player in a manner external to the network. In order to notify you, the person hosting the game might send you electronic mail, call you, or leave a sticky note on your computer telling you what game the port is on and what time to join. When you use the `NSpProtocol_CreateIP` function, you can specify the default port your game is hosted on. You can then specify the same port as the default port to use when joining a game.

inMaxRTT

The maximum round-trip time (RTT) allowed for new players. Pass 0 if you do not wish to have round-trip time checked. This does not guarantee that RTT will remain at the same level when the player joins. RTT is specified in milliseconds.

inMinThruput

The minimum throughput required of any prospective entrant into the game. Pass 0 if you do not wish to have throughput checked. This does not guarantee that throughput will remain at the same level when the player joins. Throughput is measured in bytes per second.

Return Value
A reference to the created protocol, or `NULL` if there was an error in specifying the protocol.

Discussion
Use this function if you wish to preconfigure the TCP/IP protocol before calling `NSpDoModalHostDialog` (page 76) or if you want to host the game programmatically.

Note that NetSprocket creates both TCP and UDP endpoints. System messages and messages with the Registered flag set are sent using TCP; all others are sent using UDP.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpProtocol_Dispose

Deletes a protocol reference.

Unsupported

```
void NSpProtocol_Dispose (
    NSpProtocolReference inProtocolRef
);
```

Parameters

inProtocolRef
An opaque reference to the protocol being deleted.

Return Value

Discussion
You should use this function to delete a protocol reference you created (for example, by calling `NSpProtocol_CreateIP` (page 104)).

Note that if you have added a protocol reference to a protocol list, the list owns the memory associated with the protocol reference and will delete it when the list is deleted.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpProtocol_ExtractDefinitionString

Unsupported

```
OSStatus NSpProtocol_ExtractDefinitionString (
    NSpProtocolReference inProtocolRef,
    char *outDefinitionString
);
```

Parameters

inProtocolRef

Return Value

Discussion

Version Notes

Declared In
NetSprocket.h

NSpProtocol_New

Unsupported

```
OSStatus NSpProtocol_New (
    const char *inDefinitionString,
    NSpProtocolReference *outReference
);
```

Parameters

outReference

Return Value

Discussion

Version Notes

Declared In
NetSprocket.h

NSpReleaseAddressReference

Releases memory associated with an address reference allocated by NetSprocket.

Unsupported

```
void NSpReleaseAddressReference (
    NSpAddressReference inAddress
);
```

Parameters

inAddress

A valid `NSpAddressReference` returned from `NSpDoModalJoinDialog` or `NSpConvertOTAddrToAddressReference`.

Return Value

Discussion
For efficient memory management, you should call `NSpReleaseAddressReference` when your game no longer needs an address reference.

You should only call this function to release address references that NetSprocket obtains on your behalf, such as when calling the function `NSpDoModalJoinDialog` (page 77). Address references obtained by other means must be disposed by other means. For example, to release an address reference converted from an `OTAddress`, you should release the memory associated with the address by calling `DisposePtr`.

Version Notes
Introduced with NetSprocket 1.0.

Declared In
`NetSprocket.h`

NSpSetConnectTimeout

Sets the timeout period to create a new network connection.

Unsupported

```
void NSpSetConnectTimeout (
    UInt32 inSeconds
);
```

Parameters

inSeconds

The timeout period in seconds. If you pass 0, then NetSprocket will use the default TCP timeout of 4 minutes.

Return Value

Discussion
If the timeout exceeds the limit set by this function, then NetSprocket will stop trying to create a connection. This timeout period is applies only to the game making the call.

Version Notes
Introduced with NetSprocket 1.7.

Declared In
`NetSprocket.h`

SSpConfigureSpeakerSetup

Unsupported

```
OSStatus SSpConfigureSpeakerSetup (
    SSpEventProcPtr inEventProcPtr
);
```

Parameters

inEventProcPtr

Return Value

Discussion

Version Notes

Declared In
`SoundSprocket.h`

SSpGetCPULoadLimit

Unsupported

```
OSStatus SSpGetCPULoadLimit (
    UInt32 *outCPULoadLimit
);
```

Parameters

outCPULoadLimit

Return Value

Discussion

Version Notes

Declared In
SoundSprocket.h

SSpListener_Dispose

Unsupported

```
OSStatus SSpListener_Dispose (
    SSpListenerReference inListenerReference
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In
SoundSprocket.h

SSpListener_GetActualVelocity

Unsupported

```
OSStatus SSpListener_GetActualVelocity (
    SSpListenerReference inListenerReference,
    TQ3Vector3D *outVelocity
);
```

Parameters

inListenerReference

outVelocity

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetActualVelocityfv

Unsupported

```
OSStatus SSpListener_GetActualVelocityfv (
    SSpListenerReference inListenerReference,
    float *outVelocity
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetCameraPlacement

Unsupported

```
OSStatus SSpListener_GetCameraPlacement (
    SSpListenerReference inListenerReference,
    TQ3CameraPlacement *outCameraPlacement
);
```

Parameters

inListenerReference

outCameraPlacement

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetCameraPlacementfv

Unsupported

```
OSStatus SSpListener_GetCameraPlacementfv (
    SSpListenerReference inListenerReference,
    float *outCameraPlacement,
    float *outPointOfInterest,
    float *outUpVector
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetMedium

Unsupported

```
OSStatus SSpListener_GetMedium (
    SSpListenerReference inListenerReference,
    UInt32 *outMedium,
    float *outHumidity
);
```

Parameters

inListenerReference

outMedium

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetMetersPerUnit

Unsupported

```
OSStatus SSpListener_GetMetersPerUnit (
    SSpListenerReference inListenerReference,
    float *outMetersPerUnit
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetOrientation

Unsupported

```
OSStatus SSpListener_GetOrientation (
    SSpListenerReference inListenerReference,
    TQ3Vector3D *outOrientation
);
```

Parameters

inListenerReference

outOrientation

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetOrientationfv

Unsupported

```
OSStatus SSpListener_GetOrientationfv (
    SSpListenerReference inListenerReference,
    float *outOrientation
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetPosition

Unsupported

```
OSStatus SSpListener_GetPosition (
    SSpListenerReference inListenerReference,
    TQ3Point3D *outPosition
);
```

Parameters

inListenerReference

outPosition

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetPositionfv

Unsupported

```
OSStatus SSpListener_GetPositionfv (
    SSpListenerReference inListenerReference,
    float *outPosition
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetReverb

Unsupported

```
OSStatus SSpListener_GetReverb (
    SSpListenerReference inListenerReference,
    float *outRoomSize,
    float *outRoomReflectivity,
    float *outReverbAttenuation
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetTransform

Unsupported

```
OSStatus SSpListener_GetTransform (
    SSpListenerReference inListenerReference,
    TQ3Matrix4x4 *outTransform
);
```

Parameters

inListenerReference

outTransform

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetTransformfv

Unsupported

```
OSStatus SSpListener_GetTransformfv (
    SSpListenerReference inListenerReference,
    float *outTransform
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetUpVector

Unsupported

```
OSStatus SSpListener_GetUpVector (
    SSpListenerReference inListenerReference,
    TQ3Vector3D *outUpVector
);
```

Parameters

inListenerReference

outUpVector

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetUpVectorfv

Unsupported

```
OSStatus SSpListener_GetUpVectorfv (
    SSpListenerReference inListenerReference,
    float *outUpVector
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetVelocity

Unsupported

```
OSStatus SSpListener_GetVelocity (
    SSpListenerReference inListenerReference,
    TQ3Vector3D *outVelocity
);
```

Parameters

inListenerReference

outVelocity

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_GetVelocityfv

Unsupported

```
OSStatus SSpListener_GetVelocityfv (
    SSpListenerReference inListenerReference,
    float *outVelocity
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_New

Unsupported

```
OSStatus SSpListener_New (
    SSpListenerReference *outListenerReference
);
```

Parameters

outListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetCameraPlacement

Unsupported

```
OSStatus SSpListener_SetCameraPlacement (
    SSpListenerReference inListenerReference,
    const TQ3CameraPlacement *inCameraPlacement
);
```

Parameters

inListenerReference

inCameraPlacement

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetCameraPlacementfv

Unsupported

```
OSStatus SSpListener_SetCameraPlacementfv (
    SSpListenerReference inListenerReference,
    const float *inCameraLocation,
    const float *inPointOfInterest,
    const float *inUpVector
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetMedium

Unsupported

```
OSStatus SSpListener_SetMedium (
    SSpListenerReference inListenerReference,
    UInt32 inMedium,
    float inHumidity
);
```

Parameters

inListenerReference

inMedium

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetMetersPerUnit

Unsupported

```
OSStatus SSpListener_SetMetersPerUnit (
    SSpListenerReference inListenerReference,
    float inMetersPerUnit
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetOrientation

Unsupported

```
OSStatus SSpListener_SetOrientation (
    SSpListenerReference inListenerReference,
    const TQ3Vector3D *inOrientation
);
```

Parameters

inListenerReference

inOrientation

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetOrientation3f

Unsupported

```
OSStatus SSpListener_SetOrientation3f (
    SSpListenerReference inListenerReference,
    float inX,
    float inY,
    float inZ
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetOrientationfv

Unsupported

```
OSStatus SSpListener_SetOrientationfv (
    SSpListenerReference inListenerReference,
    const float *inOrientation
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetPosition

Unsupported

```
OSStatus SSpListener_SetPosition (
    SSpListenerReference inListenerReference,
    const TQ3Point3D *inPosition
);
```

Parameters

inListenerReference

inPosition

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetPosition3f

Unsupported

```
OSStatus SSpListener_SetPosition3f (
    SSpListenerReference inListenerReference,
    float inX,
    float inY,
    float inZ
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetPositionfv

Unsupported

```
OSStatus SSpListener_SetPositionfv (
    SSpListenerReference inListenerReference,
    const float *inPosition
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetReverb

Unsupported

```
OSStatus SSpListener_SetReverb (
    SSpListenerReference inListenerReference,
    float inRoomSize,
    float inRoomReflectivity,
    float inReverbAttenuation
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetTransform

Unsupported

```
OSStatus SSpListener_SetTransform (
    SSpListenerReference inListenerReference,
    const TQ3Matrix4x4 *inTransform
);
```

Parameters

inListenerReference

inTransform

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetTransformfv

Unsupported

```
OSStatus SSpListener_SetTransformfv (
    SSpListenerReference inListenerReference,
    const float *inTransform
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetUpVector

Unsupported

```
OSStatus SSpListener_SetUpVector (
    SSpListenerReference inListenerReference,
    const TQ3Vector3D *inUpVector
);
```

Parameters

inListenerReference

inUpVector

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetUpVector3f

Unsupported

```
OSStatus SSpListener_SetUpVector3f (
    SSpListenerReference inListenerReference,
    float inX,
    float inY,
    float inZ
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetUpVectorfv

Unsupported

```
OSStatus SSpListener_SetUpVectorfv (
    SSpListenerReference inListenerReference,
    const float *inUpVector
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetVelocity

Unsupported

```
OSStatus SSpListener_SetVelocity (
    SSpListenerReference inListenerReference,
    const TQ3Vector3D *inVelocity
);
```

Parameters

inListenerReference

inVelocity

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetVelocity3f

Unsupported

```
OSStatus SSpListener_SetVelocity3f (
    SSpListenerReference inListenerReference,
    float inX,
    float inY,
    float inZ
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpListener_SetVelocityfv

Unsupported

```
OSStatus SSpListener_SetVelocityfv (
    SSpListenerReference inListenerReference,
    const float *inVelocity
);
```

Parameters

inListenerReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_CalcLocalization

Unsupported

```
OSStatus SSpSource_CalcLocalization (
    SSpSourceReference inSourceReference,
    SSpListenerReference inListenerReference,
    SSpLocalizationData *out3DInfo
);
```

Parameters

inSourceReference

inListenerReference

out3DInfo

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_Dispose

Unsupported

```
OSStatus SSpSource_Dispose (
    SSpSourceReference inSourceReference
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetActualVelocity

Unsupported

```
OSStatus SSpSource_GetActualVelocity (
    SSpSourceReference inSourceReference,
    TQ3Vector3D *outVelocity
);
```

Parameters

inSourceReference

outVelocity

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetActualVelocityfv

Unsupported

```
OSStatus SSpSource_GetActualVelocityfv (
    SSpSourceReference inSourceReference,
    float *outVelocity
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetAngularAttenuation

Unsupported

```
OSStatus SSpSource_GetAngularAttenuation (
    SSpSourceReference inSourceReference,
    float *outConeAngle,
    float *outConeAttenuation
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetCameraPlacement

Unsupported

```
OSStatus SSpSource_GetCameraPlacement (
    SSpSourceReference inSourceReference,
    TQ3CameraPlacement *outCameraPlacement
);
```

Parameters

inSourceReference

outCameraPlacement

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetCameraPlacementfv

Unsupported

```
OSStatus SSpSource_GetCameraPlacementfv (
    SSpSourceReference inSourceReference,
    float *outCameraPlacement,
    float *outPointOfInterest,
    float *outUpVector
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetCPULoad

Unsupported

```
OSStatus SSpSource_GetCPULoad (
    SSpSourceReference inSourceReference,
    UInt32 *outCPULoad
);
```

Parameters

inSourceReference

outCPULoad

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetMode

Unsupported

```
OSStatus SSpSource_GetMode (
    SSpSourceReference inSourceReference,
    UInt32 *outMode
);
```

Parameters

inSourceReference

outMode

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetOrientation

Unsupported

```
OSStatus SSpSource_GetOrientation (
    SSpSourceReference inSourceReference,
    TQ3Vector3D *outOrientation
);
```

Parameters

inSourceReference

outOrientation

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetOrientationfv

Unsupported

```
OSStatus SSpSource_GetOrientationfv (
    SSpSourceReference inSourceReference,
    float *outOrientation
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetPosition

Unsupported

```
OSStatus SSpSource_GetPosition (
    SSpSourceReference inSourceReference,
    TQ3Point3D *outPosition
);
```

Parameters

inSourceReference

outPosition

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetPositionfv

Unsupported

```
OSStatus SSpSource_GetPositionfv (
    SSpSourceReference inSourceReference,
    float *outPosition
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetReferenceDistance

Unsupported

```
OSStatus SSpSource_GetReferenceDistance (
    SSpSourceReference inSourceReference,
    float *outReferenceDistance
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetSize

Unsupported

```
OSStatus SSpSource_GetSize (
    SSpSourceReference inSourceReference,
    float *outLength,
    float *outWidth,
    float *outHeight
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetTransform

Unsupported

```
OSStatus SSpSource_GetTransform (
    SSpSourceReference inSourceReference,
    TQ3Matrix4x4 *outTransform
);
```

Parameters

inSourceReference

outTransform

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetTransformfv

Unsupported

```
OSStatus SSpSource_GetTransformfv (
    SSpSourceReference inSourceReference,
    float *outTransform
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetUpVector

Unsupported

```
OSStatus SSpSource_GetUpVector (
    SSpSourceReference inSourceReference,
    TQ3Vector3D *outUpVector
);
```

Parameters

inSourceReference

outUpVector

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetUpVectorfv

Unsupported

```
OSStatus SSpSource_GetUpVectorfv (
    SSpSourceReference inSourceReference,
    float *outUpVector
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetVelocity

Unsupported

```
OSStatus SSpSource_GetVelocity (
    SSpSourceReference inSourceReference,
    TQ3Vector3D *outVelocity
);
```

Parameters

inSourceReference

outVelocity

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_GetVelocityfv

Unsupported

```
OSStatus SSpSource_GetVelocityfv (
    SSpSourceReference inSourceReference,
    float *outVelocity
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_New

Unsupported

```
OSStatus SSpSource_New (
    SSpSourceReference *outSourceReference
);
```

Parameters

outSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetAngularAttenuation

Unsupported

```
OSStatus SSpSource_SetAngularAttenuation (
    SSpSourceReference inSourceReference,
    float inConeAngle,
    float inConeAttenuation
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetCameraPlacement

Unsupported

```
OSStatus SSpSource_SetCameraPlacement (
    SSpSourceReference inSourceReference,
    const TQ3CameraPlacement *inCameraPlacement
);
```

Parameters

inSourceReference

inCameraPlacement

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetCameraPlacementfv

Unsupported

```
OSStatus SSpSource_SetCameraPlacementfv (
    SSpSourceReference inSourceReference,
    const float *inCameraLocation,
    const float *inPointOfInterest,
    const float *inUpVector
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetCPULoad

Unsupported

```
OSStatus SSpSource_SetCPULoad (
    SSpSourceReference inSourceReference,
    UInt32 inCPULoad
);
```

Parameters

inSourceReference

inCPULoad

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetMode

Unsupported

```
OSStatus SSpSource_SetMode (
    SSpSourceReference inSourceReference,
    UInt32 inMode
);
```

Parameters

inSourceReference

inMode

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetOrientation

Unsupported

```
OSStatus SSpSource_SetOrientation (
    SSpSourceReference inSourceReference,
    const TQ3Vector3D *inOrientation
);
```

Parameters

inSourceReference

inOrientation

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetOrientation3f

Unsupported

```
OSStatus SSpSource_SetOrientation3f (
    SSpSourceReference inSourceReference,
    float inX,
    float inY,
    float inZ
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetOrientationfv

Unsupported

```
OSStatus SSpSource_SetOrientationfv (
    SSpSourceReference inSourceReference,
    const float *inOrientation
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetPosition

Unsupported

```
OSStatus SSpSource_SetPosition (
    SSpSourceReference inSourceReference,
    const TQ3Point3D *inPosition
);
```

Parameters

inSourceReference

inPosition

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetPosition3f

Unsupported

```
OSStatus SSpSource_SetPosition3f (
    SSpSourceReference inSourceReference,
    float inX,
    float inY,
    float inZ
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetPositionfv

Unsupported

```
OSStatus SSpSource_SetPositionfv (
    SSpSourceReference inSourceReference,
    const float *inPosition
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetReferenceDistance

Unsupported

```
OSStatus SSpSource_SetReferenceDistance (
    SSpSourceReference inSourceReference,
    float inReferenceDistance
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetSize

Unsupported

```
OSStatus SSpSource_SetSize (
    SSpSourceReference inSourceReference,
    float inLength,
    float inWidth,
    float inHeight
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetTransform

Unsupported

```
OSStatus SSpSource_SetTransform (
    SSpSourceReference inSourceReference,
    const TQ3Matrix4x4 *inTransform
);
```

Parameters

inSourceReference

inTransform

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetTransformfv

Unsupported

```
OSStatus SSpSource_SetTransformfv (
    SSpSourceReference inSourceReference,
    const float *inTransform
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetUpVector

Unsupported

```
OSStatus SSpSource_SetUpVector (
    SSpSourceReference inSourceReference,
    const TQ3Vector3D *inUpVector
);
```

Parameters

inSourceReference

inUpVector

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetUpVector3f

Unsupported

```
OSStatus SSpSource_SetUpVector3f (
    SSpSourceReference inSourceReference,
    float inX,
    float inY,
    float inZ
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetUpVectorfv

Unsupported

```
OSStatus SSpSource_SetUpVectorfv (
    SSpSourceReference inSourceReference,
    const float *inUpVector
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetVelocity

Unsupported

```
OSStatus SSpSource_SetVelocity (
    SSpSourceReference inSourceReference,
    const TQ3Vector3D *inVelocity
);
```

Parameters

inSourceReference

inVelocity

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetVelocity3f

Unsupported

```
OSStatus SSpSource_SetVelocity3f (
    SSpSourceReference inSourceReference,
    float inX,
    float inY,
    float inZ
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

SSpSource_SetVelocityfv

Unsupported

```
OSStatus SSpSource_SetVelocityfv (
    SSpSourceReference inSourceReference,
    const float *inVelocity
);
```

Parameters

inSourceReference

Return Value

Discussion

Version Notes

Declared In

SoundSprocket.h

Callbacks

DSpBlitDoneProc

Defines a pointer to a blitting completion function. Your callback function handles any tasks required after DrawSprocket finishes blitting between buffers.

```
typedef void (*DSpBlitDoneProc) (
    DSpBlitInfo * info
);
```

If you name your function `My`, you would declare it like this:

```
void DSpBlitDoneProc (
    DSpBlitInfo * info
);
```

Parameters

info

A pointer to a data structure containing information about the completed blitting operation. See the DSpBlitInfo structure for more information.

Return Value

Discussion

DrawSprocket calls this application-defined function during calls to the functions `DSpBlit_Faster` (page 29) or `DSpBlit_Fastest` (page 29).

If you are performing multiple asynchronous blitting operations, your application-defined completion function can check the blitter information structure passed to it to determine which operation was completed.

Version Notes

Introduced with DrawSprocket 1.1

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DrawSprocket.h`

DSpEventProcPtr

Defines a pointer to an event-handling callback function. Your callback function handles events that occur during calls to the function `DSpUserSelectContext`.

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

If you name your function `MyDSpEventProc`, you would declare it like this:

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

Parameters

inEvent

A pointer to an event record that describes the event that occurred.

Return Value

If your function handled the event, it should return `true`; otherwise it should return `false`.

Discussion

When calling the function `DSpUserSelectContext` (page 38) , you must designate this application-defined function to handle events (such as update events) that may occur while the configuration window is active.

Version Notes

Introduced with DrawSprocket 1.0.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DrawSprocket.h`

GSpEventProcPtr

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

Parameters

inEvent

Return Value

Discussion

Version Notes

ISpADBDeferCallbackProcPtr

If you name your function `MyISpADBDeferCallbackProc`, you would declare it like this:

```
void MyISpADBDeferCallbackCallback (
    UInt8 adbCommand,
    void *adbBuffer,
    UInt32 refcon
);
```

Parameters

adbCommand

refcon

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_ADBReInit

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    Boolean inPostProcess
);
```

Parameters

refCon

inPostProcess

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_BeginConfiguration

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    UInt32 count,
    ISpNeed *needs
);
```

Parameters

refCon

count

needs

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_CalibrateDialog

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    Boolean *changed
);
```

Parameters

refCon

changed

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_Click

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    const EventRecord *event
);
```

Parameters

refCon

event

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_DialogItemHit

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    SInt16 itemHit
);
```

Parameters

refCon

itemHit

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_Dirty

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    Boolean *dirty
);
```

Parameters

refCon

dirty

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_Draw

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon
);
```

Parameters

refCon

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_EndConfiguration

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    Boolean accept
);
```

Parameters

refCon

accept

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_Generic

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    SInt32 flags,
    ...
```

```
);
```

Parameters

refCon

flags

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_GetCalibration

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    void *calibration,
    Size *calibrationSize
);
```

Parameters

refCon

calibrationSize

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_GetIcon

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    SInt16 *iconSuiteResourceId
);
```

Parameters

refCon

iconSuiteResourceId

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_GetSize

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    Point *minimum,
    Point *best
);
```

Parameters

refCon

minimum

best

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_GetState

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    UInt32 buflen,
    void *buffer,
    UInt32 *length
);
```

Parameters

refCon

buflen

length

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_HandleEvent

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    EventRecord *theEvent,
    Boolean *handled
);
```

Parameters

refCon

theEvent

handled

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_Hide

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon
);
```

Parameters

refCon

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_Init

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    UInt32 count,
    ISpNeed *needs,
    ISpElementReference *virtualElements,
    Boolean *used,
    OSType appCreatorCode,
    OSType subCreatorCode,
    UInt32 reserved,
    void *reserved2
);
```

Parameters

refCon

count

needs

virtualElements

used

appCreatorCode

subCreatorCode

reserved

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_InterruptTickle

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon
);
```

Parameters

refCon

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_MetaHandler

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
ISpDriverFunctionPtr_Generic MyCallback
(
    UInt32 refCon,
    ISpMetaHandlerSelector metaHandlerSelector
);
```

Parameters

refCon

metaHandlerSelector

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_SetActive

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    Boolean active
);
```

Parameters

refCon

active

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_SetCalibration

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    void *calibration,
    Size calibrationSize
);
```

Parameters

refCon

calibrationSize

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_SetState

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
```

```
    UInt32 length,
    void *buffer
);
```

Parameters

refCon

length

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_Show

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon,
    DialogRef theDialog,
    SInt16 dialogItemNumber,
    Rect *r
);
```

Parameters

refCon

theDialog

dialogItemNumber

r

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_Stop

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon
);
```

Parameters

refCon

Return Value

Discussion

Version Notes

ISpDriverFunctionPtr_Tickle

If you name your function `MyISpDriverFunction`, you would declare it like this:

```
OSStatus MyCallback (
    UInt32 refCon
);
```

Parameters

refCon

Return Value

Discussion

Version Notes

ISpDriver_CheckConfigurationPtr

If you name your function `MyISpDriver_CheckConfiguration`, you would declare it like this:

```
OSStatus MyCallback (
    Boolean *validConfiguration
);
```

Parameters

validConfiguration

Return Value

Discussion

Version Notes

ISpDriver_DisposeDevicesPtr

If you name your function `MyISpDriver_DisposeDevices`, you would declare it like this:

```
OSStatus MyCallback ();
```

Parameters

Return Value

Discussion

Version Notes

ISpDriver_FindAndLoadDevicesPtr

If you name your function `MyISpDriver_FindAndLoadDevices`, you would declare it like this:

```
OSStatus MyCallback (
    Boolean *keepDriverLoaded
);
```

Parameters

keepDriverLoaded

Return Value

Discussion

Version Notes

ISpDriver_TicklePtr

If you name your function `MyISpDriver_Tickle`, you would declare it like this:

```
void MyCallback ();
```

Parameters

Return Value

Discussion

Version Notes

ISpEventProcPtr

Defines a pointer to an event handler callback function. Your event handler processes events that occur during calls to the function `ISpConfigure`.

If you name your function `MyISpEventProc`, you would declare it like this:

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

Parameters

inEvent

A pointer to an event record that describes the event that occurred.

Return Value
If your function handled the event, it should return `True`; otherwise it should return `False`.

Discussion
Your callback function handles events that may occur while the configuration window is active.

Version Notes
Introduced with InputSprocket 1.0.

NSpCallbackProcPtr

If you name your function `MyNSpCallbackProc`, you would declare it like this:

```
void MyNSpCallbackCallback (
    NSpGameReference inGame,
    void *inContext,
    NSpEventCode inCode,
    OSStatus inStatus,
    void *inCookie
);
```

Parameters
inGame
inCode
inStatus
Return Value
Discussion
Version Notes

NSpEventProcPtr

If you name your function `MyNSpEventProc`, you would declare it like this:

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

Parameters
inEvent
Return Value
Discussion
Version Notes

NSpJoinRequestHandlerProcPtr

Defines a pointer to a join request callback function. You can use this callback function to specify your own criteria for joining a game.

If you name your function `MyNSpJoinRequestHandlerProc`, you would declare it like this:

```
Boolean MyNSpJoinRequestHandlerCallback
(
    NSpGameReference inGame,
    NSpJoinRequestMessage *inMessage,
    void *inContext,
    Str255 outReason
);
```

Parameters

inGame

An opaque reference to the game object that received the join request.

inMessage

A pointer to the join request message. This is data passed to your function by NetSprocket. It will contain the name, password, and any custom data that your game specifies.

inContext

The context pointer you passed to NetSprocket when you first installed the join request handler.

outReason

A pointer to a Pascal string that NetSprocket will allocate for you. You can use this string to send textual information to a player. For example, if you are going to deny a join request, you may send your reason for denial into `outReason`.

Return Value

A value of `true` to inform NetSprocket to allow the prospective player into the game, or `false` to deny entry based on the criteria you have established.

Discussion

This is a function that you as the game developer must provide if you are going to provide a custom join request handler. Once you have installed your join request handler, it will be called whenever a new player wishes to enter the game. Your function must return `true` or `false`, telling NetSprocket whether or not to admit the prospective player.

The purpose of the custom function is to allow more flexibility in controlling access to the game. By default, NetSprocket allows players to join the game based on the password and minimum round-trip time of the prospective player. However, you may want to restrict play to a particular network zone, or you may decide that certain levels of games may be played only by players with a previous score history.

Also note that before calling your request handler, NetSprocket will always make two checks for a prospective player. First, it will make sure that the prospective player's round-trip time meets your minimum requirements, if you have specified any. Second, it will make sure that allowing this player into the game will not exceed your maximum player count.

You should not release the message passed to this function.

Version Notes

Introduced with NetSprocket 1.0.

NSpMessageHandlerProcPtr

Defines a pointer to a message handling callback function. You can use this callback function to supply custom code for handling incoming messages.

If you name your function `MyNSpMessageHandlerProc`, you would declare it like this:

```
Boolean MyNSpMessageHandlerCallback (
    NSpGameReference inGame,
    NSpMessageHeader *inMessage,
    void *inContext
);
```

Parameters

inGame

An opaque reference to the game object that received the message.

inMessage

A pointer to the message.

inContext

The context pointer you passed in when you installed the handler.

Return Value

Discussion

Your function must handle the message and return as quickly as possible. You should not free the message, as it will be automatically freed when your function returns. If you return `true`, then NetSprocket will put the message back into the incoming message queue. When you call the `NSpMessage_Get` (page 91) function you will receive the message again. If you return `false`, the message will be deleted when your function returns. As an example, if you receive a message and you want to change part of the message or add to it, you can make a note in the message and then receive it again (by calling `NSpMessage_Get` (page 91))with the note added to the message. You can also use this as a mechanism for time stamping messages and only act on the latest messages.

You do not need to define a function of this type if you use NetSprocket in the normal event-loop mode.

Your handler must obey all the rules of interrupt-safe functions.

Version Notes

Introduced with NetSprocket 1.0.

SSpEventProcPtr

If you name your function `MySSpEventProc`, you would declare it like this:

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

Parameters

inEvent

Return Value

Discussion

Version Notes

Data Types

GSpEventProcPtr

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

Discussion

ISpADBDeferCallbackProcPtr

```
typedef void (*ISpADBDeferCallbackProcPtr) (
    UInt8 adbCommand,
    void *adbBuffer,
    UInt32 refcon
);
```

Discussion

ISpADBDeferRef

```
typedef UInt32 ISpADBDeferRef;
```

Discussion

Version Notes

ISpApplicationResourceStruct

Describes the InputSprocket features used by the application.

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

Fields

Discussion
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.

Version Notes
Introduced with InputSprocket 1.2.

ISpAxisConfigurationInfo

Holds axis element configuration information.

```
struct ISpAxisConfigurationInfo {
    Boolean symetricAxis;
};
```

Fields

Discussion
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.

Version Notes
Introduced with InputSprocket 1.0.

ISpAxisData

```
typedef UInt32 ISpAxisData;
```

Discussion

Version Notes

ISpButtonConfigurationInfo

Holds button element configuration information.

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

Fields

Discussion
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.

Version Notes
Introduced with InputSprocket 1.0.

ISpButtonData

Specify information for data of kind `kButtonData`.

```
typedef UInt32 ISpButtonData;
enum {
    kISpButtonUp = ,
    kISpButtonDown = 1
};
```

Discussion
Use the `ISpButtonConfigurationInfo` structure for help in interpreting the data.

Version Notes
Introduced with InputSprocket 1.0.

ISpDeltaConfigurationInfo

Holds delta element configuration information.

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

Fields

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

Version Notes
Introduced with InputSprocket 1.2.

ISpDeltaData

```
typedef Fixed ISpDeltaData;
```

Discussion

Version Notes

ISpDeviceClass

Represents a class of input devices.

```
typedef OSType ISpDeviceClass;
```

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

Version Notes
Introduced with InputSprocket 1.0.

ISpDeviceDefinition

Describes an input device.

```
struct ISpDeviceDefinition {
    Str63 deviceName;
    ISpDeviceClass theDeviceClass;
    ISpDeviceIdentifier theDeviceIdentifier;
    UInt32 permanentID;
    UInt32 flags;
    UInt32 reserved1;
    UInt32 reserved2;
    UInt32 reserved3;
};
```

Fields

Discussion
A device definition structure provides all the information about the input device available within the system.

Version Notes
Introduced with InputSprocket 1.0.

ISpDeviceIdentifier

Represents a physical input device.

```
typedef OSType ISpDeviceIdentifier;
```

Discussion
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.

ISpDeviceReference

Represents an input device.

```
typedef struct * ISpDeviceReference;
```

Discussion

Version Notes
Introduced with InputSprocket 1.0.

ISpDPadConfigurationInfo

Holds directional pad element configuration information.

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

Fields

Discussion
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.

Version Notes
Introduced with InputSprocket 1.0.

ISpDPadData

Specify information for data of kind `kElementData`.

```
typedef UInt32 ISpDPadData;
enum {
    kISpPadIdle = ,
    kISpPadLeft = 1,
    kISpPadUpLeft = 2,
    kISpPadUp = 3,
    kISpPadUpRight = 4,
    kISpPadRight = 5,
    kISpPadDownRight = 6,
    kISpPadDown = 7,
    kISpPadDownLeft = 8
};
```

Discussion
Use the `ISpDPadConfigurationInfo` structure for help in interpreting the data.

Version Notes
Introduced with InputSprocket 1.0.

ISpDriver_CheckConfigurationPtr

```
typedef OSStatus (*ISpDriver_CheckConfigurationPtr)
(
    Boolean *validConfiguration
);
```

Discussion

ISpDriver_DisposeDevicesPtr

```
typedef OSStatus (*ISpDriver_DisposeDevicesPtr) (

);
```

Discussion

ISpDriver_FindAndLoadDevicesPtr

```
typedef OSStatus (*ISpDriver_FindAndLoadDevicesPtr)
(
    Boolean *keepDriverLoaded
);
```

Discussion

ISpDriver_TicklePtr

```
typedef void (*ISpDriver_TicklePtr) (
```

```
);
```

Discussion

ISpDriverFunctionPtr_ADBReInit

```
typedef OSStatus (*ISpDriverFunctionPtr_ADBReInit)
(
    UInt32 refCon,
    Boolean inPostProcess
);
```

Discussion

ISpDriverFunctionPtr_BeginConfiguration

```
typedef OSStatus (*ISpDriverFunctionPtr_BeginConfiguration)
(
    UInt32 refCon,
    UInt32 count,
    ISpNeed *needs
);
```

Discussion

ISpDriverFunctionPtr_CalibrateDialog

```
typedef OSStatus (*ISpDriverFunctionPtr_CalibrateDialog)
(
    UInt32 refCon,
    Boolean *changed
);
```

Discussion

ISpDriverFunctionPtr_Click

```
typedef OSStatus (*ISpDriverFunctionPtr_Click) (
    UInt32 refCon,
    const EventRecord *event
);
```

Discussion

ISpDriverFunctionPtr_DialogItemHit

```
typedef OSStatus (*ISpDriverFunctionPtr_DialogItemHit)
(
    UInt32 refCon,
```

```
    SInt16 itemHit
);
```

Discussion

ISpDriverFunctionPtr_Dirty

```
typedef OSStatus (*ISpDriverFunctionPtr_Dirty) (
    UInt32 refCon,
    Boolean *dirty
);
```

Discussion

ISpDriverFunctionPtr_Draw

```
typedef OSStatus (*ISpDriverFunctionPtr_Draw) (
    UInt32 refCon
);
```

Discussion

ISpDriverFunctionPtr_EndConfiguration

```
typedef OSStatus (*ISpDriverFunctionPtr_EndConfiguration)
(
    UInt32 refCon,
    Boolean accept
);
```

Discussion

ISpDriverFunctionPtr_Generic

```
typedef OSStatus (*ISpDriverFunctionPtr_Generic) (
    UInt32 refCon,
    SInt32 flags
);
```

Discussion

ISpDriverFunctionPtr_GetCalibration

```
typedef OSStatus (*ISpDriverFunctionPtr_GetCalibration)
(
    UInt32 refCon,
    void *calibration,
    Size *calibrationSize
);
```

Discussion

ISpDriverFunctionPtr_GetIcon

```
typedef OSStatus (*ISpDriverFunctionPtr_GetIcon) (
    UInt32 refCon,
    SInt16 *iconSuiteResourceId
);
```

Discussion

ISpDriverFunctionPtr_GetSize

```
typedef OSStatus (*ISpDriverFunctionPtr_GetSize) (
    UInt32 refCon,
    Point *minimum,
    Point *best
);
```

Discussion

ISpDriverFunctionPtr_GetState

```
typedef OSStatus (*ISpDriverFunctionPtr_GetState) (
    UInt32 refCon,
    UInt32 buflen,
    void *buffer,
    UInt32 *length
);
```

Discussion

ISpDriverFunctionPtr_HandleEvent

```
typedef OSStatus (*ISpDriverFunctionPtr_HandleEvent)
(
    UInt32 refCon,
    EventRecord *theEvent,
    Boolean *handled
);
```

Discussion

ISpDriverFunctionPtr_Hide

```
typedef OSStatus (*ISpDriverFunctionPtr_Hide) (
    UInt32 refCon
);
```

Discussion

ISpDriverFunctionPtr_Init

```
typedef OSStatus (*ISpDriverFunctionPtr_Init) (
    UInt32 refCon,
    UInt32 count,
    ISpNeed *needs,
    ISpElementReference *virtualElements,
    Boolean *used,
    OSType appCreatorCode,
    OSType subCreatorCode,
    UInt32 reserved,
    void *reserved2
);
```

Discussion

ISpDriverFunctionPtr_InterruptTickle

```
typedef OSStatus (*ISpDriverFunctionPtr_InterruptTickle)
(
    UInt32 refCon
);
```

Discussion

ISpDriverFunctionPtr_MetaHandler

```
typedef ISpDriverFunctionPtr_Generic (*ISpDriverFunctionPtr_MetaHandler)
(
    UInt32 refCon,
    ISpMetaHandlerSelector metaHandlerSelector
);
```

Discussion

ISpDriverFunctionPtr_SetActive

```
typedef OSStatus (*ISpDriverFunctionPtr_SetActive)
(
    UInt32 refCon,
    Boolean active
);
```

Discussion

ISpDriverFunctionPtr_SetCalibration

```
typedef OSStatus (*ISpDriverFunctionPtr_SetCalibration)
(
    UInt32 refCon,
    void *calibration,
    Size calibrationSize
);
```

Discussion

ISpDriverFunctionPtr_SetState

```
typedef OSStatus (*ISpDriverFunctionPtr_SetState) (
    UInt32 refCon,
    UInt32 length,
    void *buffer
);
```

Discussion

ISpDriverFunctionPtr_Show

```
typedef OSStatus (*ISpDriverFunctionPtr_Show) (
    UInt32 refCon,
    DialogRef theDialog,
    SInt16 dialogItemNumber,
    Rect *r
);
```

Discussion

ISpDriverFunctionPtr_Stop

```
typedef OSStatus (*ISpDriverFunctionPtr_Stop) (
    UInt32 refCon
);
```

Discussion

ISpDriverFunctionPtr_Tickle

```
typedef OSStatus (*ISpDriverFunctionPtr_Tickle) (
    UInt32 refCon
);
```

Discussion

ISpElementDefinitionStruct

```
struct ISpElementDefinitionStruct {
```

```
    ISpDeviceReference device;
    UInt32 group;
    Str63 theString;
    ISpElementKind kind;
    ISpElementLabel theLabel;
    void *configInfo;
    UInt32 configInfoLength;
    UInt32 dataSize;
    UInt32 reserved1;
    UInt32 reserved2;
    UInt32 reserved3;
};
```

Fields

Discussion

Version Notes

ISpElementEvent

Represents an event generated by an element.

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

Fields

Discussion
The element event structure is a variable length structure that passes element event data.

Version Notes
Introduced with InputSprocket 1.0.

ISpElementEventPtr

```
typedef ISpElementEvent* ISpElementEventPtr;
```

Discussion

Version Notes

ISpElementInfo

Describes an element.

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

Fields

Discussion
The element information structure provides basic information about an element that is common to all elements, regardless of kind.

Version Notes
Introduced with InputSprocket 1.0.

ISpElementInfoPtr

```
typedef ISpElementInfo* ISpElementInfoPtr;
```

Discussion

Version Notes

ISpElementKind

Represents an element kind.

```
typedef OSType ISpElementKind;
```

Discussion

Version Notes
Introduced with InputSprocket 1.0.

ISpElementLabel

Represents an element label.

```
typedef OSType ISpElementLabel;
```

Discussion

Version Notes
Introduced with InputSprocket 1.0.

ISpElementListReference

Represents an element list.

```
typedef struct * ISpElementListReference;
```

Discussion

Version Notes
Introduced with InputSprocket 1.0.

ISpElementReference

Represents a virtual element.

```
typedef struct * ISpElementReference;
```

Discussion

Version Notes
Introduced with InputSprocket 1.0.

ISpEventProcPtr

Defines a pointer to an event handler callback function.

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

Discussion

ISpMetaHandlerSelector

```
typedef UInt32 ISpMetaHandlerSelector;
enum {
    kISpSelector_Init = 1,
    kISpSelector_Stop = 2,
    kISpSelector_GetSize = 3,
    kISpSelector_HandleEvent = 4,
    kISpSelector_Show = 5,
    kISpSelector_Hide = 6,
    kISpSelector_BeginConfiguration = 7,
    kISpSelector_EndConfiguration = 8,
    kISpSelector_GetIcon = 9,
    kISpSelector_GetState = 10,
    kISpSelector_SetState = 11,
    kISpSelector_Dirty = 12,
    kISpSelector_SetActive = 13,
    kISpSelector_DialogItemHit = 14,
    kISpSelector_Tickle = 15,
    kISpSelector_InterruptTickle = 16,
    kISpSelector_Draw = 17,
    kISpSelector_Click = 18,
    kISpSelector_ADBReInit = 19,
    kISpSelector_GetCalibration = 20,
    kISpSelector_SetCalibration = 21,
    kISpSelector_CalibrateDialog = 22
};
```

Discussion

Version Notes

ISpMovementConfigurationInfo

Holds movement element configuration information.

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

Fields

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

Version Notes
Introduced with InputSprocket 1.1.

ISpMovementData

Holds movement element data .

```
struct ISpMovementData {
    ISpAxisData xAxis;
    ISpAxisData yAxis;
    ISpDPadData direction;
};
```

Fields

Discussion
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.

Version Notes
Introduced with InputSprocket 1.0.

ISpNeed

Describes the type of element required for a game input.

```
struct ISpNeed {
    Str63 name;
    SInt16 iconSuiteResourceId;
    UInt8 playerNum;
    UInt8 group;
    ISpElementKind theKind;
    ISpElementLabel theLabel;
    ISpNeedFlagBits flags;
    UInt32 reserved1;
    UInt32 reserved2;
    UInt32 reserved3;
};
```

Fields

Discussion
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.

Version Notes
This version of the need structure introduced with InputSprocket 1.2.

ISpNeedFlagBits

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

```
typedef UInt32 ISpNeedFlagBits;
```

Discussion

Version Notes
Introduced with InputSprocket 1.0.

NSpAddPlayerToGroupMessage

Describes a message indicating that NetSprocket added a player to a group.

```
struct NSpAddPlayerToGroupMessage {
    NSpMessageHeader header;
    NSpGroupID group;
    NSpPlayerID player;
};
```

Fields

Discussion
NetSprocket uses the `NSpAddPlayerToGroupMessage` structure to send a message to all players when a player is added to a group. It indicates player added to group messages by passing the constant `kNSpPlayerAddedToGroup` in the `what` field of the `NSpMessageHeader` (page 181) structure. Note that NetSprocket handles this message internally unless you had specified a custom message handler, in which case you can interpret the message in your handler and take any desired actions.

Version Notes
Introduced with NetSprocket 1.7.

NSpAddressReference

Represents a network address.

```
typedef struct * NSpAddressReference;
```

Discussion
You use the address reference to manipulate protocol references. You obtain an address reference by calling `NSpDoModalJoinDialog` (page 77) or by converting an Open Transport `OTAddress`.

Version Notes
Introduced with NetSprocket 1.0.

NSpCallbackProcPtr

```
typedef void (*NSpCallbackProcPtr) (
    NSpGameReference inGame,
    void *inContext,
    NSpEventCode inCode,
    OSStatus inStatus,
    void *inCookie
);
```

Discussion

NSpCreateGroupMessage

Describes a message indicating that NetSprocket created a group.

```
struct NSpCreateGroupMessage {
    NSpMessageHeader header;
    NSpGroupID groupID;
    NSpPlayerID requestingPlayer;
};
```

Fields

Discussion
NetSprocket uses the `NSpCreateGroupMessage` structure to send a message to all players when a group is created. It indicates group created messages by passing the constant `kNSpGroupCreated` in the `what` field of the `NSpMessageHeader` (page 181) structure. Note that NetSprocket handles this message internally unless you had specified a custom message handler, in which case you can interpret the message in your handler and take any desired actions.

Version Notes
Introduced with NetSprocket 1.7.

NSpDeleteGroupMessage

Describes a message indicating that NetSprocket deleted a group.

```
struct NSpDeleteGroupMessage {
    NSpMessageHeader header;
    NSpGroupID groupID;
    NSpPlayerID requestingPlayer;
};
```

Fields

Discussion
NetSprocket uses the `NSpDeleteGroupMessage` structure to send a message to all players when a group is removed. It indicates group deleted messages by passing the constant `kNSpGroupDeleted` in the `what` field of the `NSpMessageHeader` (page 181) structure. Note that NetSprocket handles this message internally unless you had specified a custom message handler, in which case you can interpret the message in your handler and take any desired actions.

Version Notes
Introduced with NetSprocket 1.7.

NSpErrorMessage

Describes an error message.

```
struct NSpErrorMessage {
    NSpMessageHeader header;
    OSStatus error;
};
```

Fields

Discussion

The error message structure is a standard NetSprocket message you receive when extreme error conditions occur in NetSprocket. This can occur when functions fail or when network failures among players are detected by NetSprocket in the course of the game. NetSprocket indicates error messages by passing the constant `kNSpError` in the `what` field of the `NSpMessageHeader` (page 181) structure. The error message structure is defined by the `NSpErrorMessage` data type.

Version Notes

Introduced with NetSprocket 1.0.

NSpEventCode

```
typedef SInt32 NSpEventCode;
```

Discussion

Version Notes

NSpEventProcPtr

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

Discussion

NSpFlags

Identifies game options.

```
typedef SInt32 NSpFlags;
```

Discussion

A number of NetSprocket functions (such as `NSpGame_Host` (page 80) and `NSpGame_Join` (page 82)) allow you to specify options by passing constants of type `NSpFlags`.

Version Notes

Introduced with NetSprocket 1.0.

NSpGameID

Identifies a game on the network.

```
typedef SInt32 NSpGameID;
```

Discussion
When calling the function `NSpInitialize` (page 88) , you must specify a unique ID that NetSprocket will use to keep track of your game on the network.

Version Notes
Introduced with NetSprocket 1.0.

NSpGameInfo

Contains information about the current game.

```
struct NSpGameInfo {
    UInt32 maxPlayers;
    UInt32 currentPlayers;
    UInt32 currentGroups;
    NSpTopology topology;
    UInt32 reserved;
    Str31 name;
    Str31 password;
};
```

Fields

Discussion
Basic information about the game is organized in the game information structure. You can use this structure to maintain and obtain basic information about key elements in the game, including information about players, groups, and topology.

Version Notes
Introduced with NetSprocket 1.0.

NSpGameReference

Identifies a game to the NetSprocket library.

```
typedef struct * NSpGameReference;
```

Discussion
You can obtain a game reference by calling the function `NSpGame_Host` (page 80) or `NSpGame_Join` (page 82).

Version Notes
Introduced with NetSprocket 1.0.

NSpGameTerminatedMessage

Describes a message indicating that a game has ended.

```
struct NSpGameTerminatedMessage {
    NSpMessageHeader header;
};
```

Fields

Discussion
NetSprocket uses the game terminated message structure to send a message to all players when a game in progress has ended. This is an advisory message that contains no additional information. NetSprocket indicates game terminated messages by passing the constant `kNSpGameTerminated` in the `what` field of the `NSpMessageHeader` (page 181) structure.

Version Notes
Introduced with NetSprocket 1.0.

NSpGroupEnumeration

Lists all groups in a given game.

```
struct NSpGroupEnumeration {
    UInt32 count;
    NSpGroupInfoPtr groups[1];
};
typedef NSpGroupEnumeration* NSpGroupEnumeration;
```

Fields

Discussion
You use the group enumeration structure to obtain a list of all the groups currently in the game. In addition to the number of groups currently in the game, it contains an array of pointers to the group information structures.

Version Notes
Introduced with NetSprocket 1.0.

NSpGroupEnumerationPtr

```
typedef NSpGroupEnumeration* NSpGroupEnumerationPtr;
```

Discussion

Version Notes

NSpGroupID

Identifies an arbitrary group of players.

```
typedef NSpPlayerID NSpGroupID;
```

Discussion
NetSprocket allows you to organize players into arbitrary groups. Each such group is identified by a group ID.

NetSprocket automatically assigns an ID to a group when you call the function `NSpGroup_New` (page 86).

Version Notes
Introduced with NetSprocket 1.0.

NSpGroupInfo

Lists all players in a given group.

```
struct NSpGroupInfo {
    NSpGroupID id;
    UInt32 playerCount;
    NSpPlayerID players[1];
};
typedef struct NSpGroupInfo* NSpGroupInfo;
```

Fields

Discussion
You use the group information structure to obtain information about each of the players in a group. It includes the number of players, along with an array of the player IDs.

Version Notes
Introduced with NetSprocket 1.0.

NSpGroupInfoPtr

```
typedef NSpGroupInfo* NSpGroupInfoPtr;
```

Discussion

Version Notes

NSpHostChangedMessage

```
struct NSpHostChangedMessage {
    NSpMessageHeader header;
    NSpPlayerID newHost;
};
```

Fields

Discussion

Version Notes

NSpJoinApprovedMessage

Describes a message indicating that a join request was approved.

```
struct NSpJoinApprovedMessage {
    NSpMessageHeader header;
};
```

Fields

Discussion
When your application is hosting a game, you can use the join approved message structure to send a message to the player who has been granted entry to the game. This is an advisory message; there are no additional information fields. NetSprocket indicates join approved messages by passing the constant `kNSpJoinApproved` in the `what` field of the `NSpMessageHeader` (page 181) structure.

Version Notes
Introduced with NetSprocket 1.0.

NSpJoinDeniedMessage

Describes a message indicating that a join request was denied.

```
struct NSpJoinDeniedMessage {
    NSpMessageHeader header;
    Str255 reason;
};
```

Fields

Discussion
When your application is hosting a game, you can send the join denied message structure to a prospective player who has been denied entry into the game. If a request to join a game is denied, subsequent calls by the player attempting to join the game will return an error from NetSprocket. NetSprocket indicates join denied messages by passing the constant `kNSpJoinDenied` in the `what` field of the `NSpMessageHeader` (page 181) structure. The game object should be deleted when a join request is denied.

Version Notes
Introduced with NetSprocket 1.0.

NSpJoinRequestHandlerProcPtr

Defines a pointer to a join request callback function.

```
typedef Boolean (*NSpJoinRequestHandlerProcPtr) (
    NSpGameReference inGame,
    NSpJoinRequestMessage *inMessage,
    void *inContext,
    Str255 outReason
);
```

Discussion

NSpJoinRequestMessage

Describes a join request message.

```
struct NSpJoinRequestMessage {
    NSpMessageHeader header;
    Str31 name;
    Str31 password;
    UInt32 theType;
    UInt32 customDataLen;
    UInt8 customData[1];
};
```

Fields

Discussion
The join request message structure is a standard NetSprocket network message you can use to notify the hosting application that a player wishes to join a game about to start or one that is in progress. NetSprocket indicates join request messages by passing the constant `kNSpJoinRequest` in the `what` field of the `NSpMessageHeader` (page 181) structure. This structure will only be passed to your application if you install a custom join request handler. You will not get this structure via the `NSpMessage_Get` (page 91) function. See the function `NSpInstallJoinRequestHandler` (page 90) for more information.

Version Notes
Introduced with NetSprocket 1.0.

NSpMessageHandlerProcPtr

Defines a pointer to a message handling callback function.

```
typedef Boolean (*NSpMessageHandlerProcPtr) (
    NSpGameReference inGame,
    NSpMessageHeader *inMessage,
    void *inContext
);
```

Discussion

NSpMessageHeader

Gives generic information about a network message.

```
struct NSpMessageHeader {
    UInt32 version;
    SInt32 what;
    NSpPlayerID from;
    NSpPlayerID toID;
    UInt32 id;
    UInt32 when;
    UInt32 messageLen;
};
```

Fields

Discussion
The most important structure in NetSprocket is the abstract message type. It is comprised of the `NSpMessageHeader` itself and is followed by custom data. The message header structure contains information about the nature of the message being delivered.

NetSprocket uses the fields of the message header to deliver your message to the specified recipients. Before you send a network message, you should fill in the `what`, `to`, and `messageLen` parameters. NetSprocket will set the remaining parameters.

Version Notes
Introduced with NetSprocket 1.0.

NSpPlayerEnumeration

Lists all players in a given game.

```
struct NSpPlayerEnumeration {
    UInt32 count;
    NSpPlayerInfoPtr playerInfo[1];
};
typedef struct NSpPlayerEnumeration* NSpPlayerEnumeration;
```

Fields

Discussion
You use the player enumeration structure to obtain a list of all the players currently in the game. It contains a count of the players, followed by pointers to each of the `playerInfo` structures.

Version Notes
Introduced with NetSprocket 1.0.

NSpPlayerEnumerationPtr

```
typedef NSpPlayerEnumeration* NSpPlayerEnumerationPtr;
```

Discussion

Version Notes

NSpPlayerID

Identifies a game player.

```
typedef SInt32 NSpPlayerID;
```

Discussion
Each player in a game has a unique player ID so NetSprocket can keep track of them on the network.

NetSprocket automatically assigns a player ID to each player who joins the game.

Version Notes
Introduced with NetSprocket 1.0.

NSpPlayerInfo

Describes an individual player.

```
struct NSpPlayerInfo {
    NSpPlayerID id;
    NSpPlayerType playerType;
    Str31 name;
    UInt32 groupCount;
    NSpGroupID groups[1];
};
typedef struct NSpPlayerInfo* NSpPlayerInfo;
```

Fields

Discussion
You use the player information structure to obtain information about each player in the game. It contains the player's ID, along with pertinent information about the player, including the groups he may belong to.

Version Notes
Introduced with NetSprocket 1.0.

NSpPlayerInfoPtr

```
typedef NSpPlayerInfo* NSpPlayerInfoPtr;
```

Discussion

Version Notes

NSpPlayerJoinedMessage

Describes a message indicating that a player joined the game.

```
struct NSpPlayerJoinedMessage {
    NSpMessageHeader header;
    UInt32 playerCount;
    NSpPlayerInfo playerInfo;
};
```

Fields

Discussion
The player joined message structure is used to send a message to all players in the game to notify them that a player has joined a game. It includes an updated count of players and the new player's data structure. NetSprocket indicates player joined messages by passing the constant `kNSpPlayerJoined` in the `what` field of the `NSpMessageHeader` (page 181) structure.

Version Notes
Introduced with NetSprocket 1.0.

NSpPlayerLeftMessage

Describes a message indicating that a player left the game.

```
struct NSpPlayerLeftMessage {
    NSpMessageHeader header;
    UInt32 playerCount;
    NSpPlayerID playerID;
    NSpPlayerName playerName;
};
```

Fields

Discussion
The player left message structure is used to send a message to all players when a player leaves a game. It includes the updated count of players and the ID of the player who has departed. NetSprocket indicates player left messages by passing the constant `kNSpPlayerLeft` in the `what` field of the `NSpMessageHeader` (page 181) structure.

Version Notes
Introduced with NetSprocket 1.0.

NSpPlayerName

Represents the name of a player.

```
typedef Str31 NSpPlayerName;
```

Discussion

Version Notes
Introduced with NetSprocket 1.0.

NSpPlayerType

Identifies a player type.

```
typedef UInt32 NSpPlayerType;
```

Discussion
Each player in a game can have a player type, which is an arbitrary classification determined by the application.

Version Notes
Introduced with NetSprocket 1.0.

NSpPlayerTypeChangedMessage

Describes a message indicating that a player type has changed.

```
struct NSpPlayerTypeChangedMessage {
    NSpMessageHeader header;
    NSpPlayerID player;
    NSpPlayerType newType;
};
```

Fields

Discussion
NetSprocket uses the `NSpPlayerTypeChangedMessage` structure to send a message indicating that a player's type has changed. It indicates player type changed messages by passing the constant `kNSpPlayerTypeChanged` in the `what` field of the `NSpMessageHeader` (page 181) structure.

Version Notes
Introduced with NetSprocket 1.7.

NSpProtocolListReference

Represents a list of protocol references.

```
typedef struct * NSpProtocolListReference;
```

Discussion
You use the protocol list reference to refer to a list of protocol references. You pass this list to the function `NSpGame_Host` (page 80) to tell NetSprocket which protocols the game is to be hosted (advertised) on.

Version Notes
Introduced with NetSprocket 1.0.

NSpProtocolReference

Represents a networking protocol.

```
typedef struct * NSpProtocolReference;
```

Discussion
You use the protocol reference to identify and configure transport protocols without having to know what the protocol actually is.

You obtain a protocol reference by calling the function `NSpDoModalHostDialog` (page 76) , `NSpProtocol_CreateAppleTalk` (page 103) , or `NSpProtocol_CreateIP` (page 104).

Version Notes
Introduced with NetSprocket 1.0.

NSpRemovePlayerFromGroupMessage

Describes a message indicating that NetSprocket removed a player from a group.

```
struct NSpRemovePlayerFromGroupMessage {
    NSpMessageHeader header;
    NSpGroupID group;
    NSpPlayerID player;
};
```

Fields

Discussion

NetSprocket uses the `NSpRemovePlayerFromGroupMessage` structure to send a message to all players when a player is removed from a group. It indicates player removed grom group messages by passing the constant `kNSpPlayerRemovedFromGroup` in the `what` field of the `NSpMessageHeader` (page 181) structure. Note that NetSprocket handles this message internally unless you had specified a custom message handler, in which case you can interpret the message in your handler and take any desired actions.

Version Notes

Introduced with NetSprocket 1.7.

NSpTopology

```
typedef UInt32 NSpTopology;
```

Discussion

Version Notes

SSpEventProcPtr

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

Discussion

SSpListenerReference

```
typedef struct * SSpListenerReference;
```

Discussion

Version Notes

SSpLocalizationData

```
struct SSpLocalizationData {
    UInt32 cpuLoad;
    UInt32 medium;
    float humidity;
    float roomSize;
    float roomReflectivity;
    float reverbAttenuation;
    UInt32 sourceMode;
    float referenceDistance;
    float coneAngleCos;
    float coneAttenuation;
    SSpLocationData currentLocation;
    UInt32 reserved0;
    UInt32 reserved1;
    UInt32 reserved2;
    UInt32 reserved3;
    UInt32 virtualSourceCount;
    SSpVirtualSourceData virtualSource[4];
};
```

Fields

Discussion

Version Notes

SSpLocationData

```
struct SSpLocationData {
    float elevation;
    float azimuth;
    float distance;
    float projectionAngle;
    float sourceVelocity;
    float listenerVelocity;
};
```

Fields

Discussion

Version Notes

SSpSourceReference

```
typedef struct * SSpSourceReference;
```

Discussion

Version Notes

SSpSpeakerSetupData

```
struct SSpSpeakerSetupData {
    UInt32 speakerKind;
    float speakerAngle;
    UInt32 reserved0;
    UInt32 reserved1;
};
```

Fields

Discussion

Version Notes

SSpVirtualSourceData

```
struct SSpVirtualSourceData {
    float attenuation;
    SSpLocationData location;
};
```

Fields

Discussion

Version Notes

Constants

Application Resource Constants

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

```
enum {
    kISpAppResFlag_UsesInputSprocket = 1,
    kISpAppResFlag_UsesISpInit = 2
};
```

Constants

`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.

Discussion

Version Notes
Introduced with InputSprocket 1.2.

Built-in Device Categories

Identify the general category of an input device.

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

Constants

`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 72) in order to operate (for example, speech recognition).

`kISpDeviceClass_Unknown`
The device is of an unknown class.

Discussion

Version Notes
Introduced with InputSprocket 1.0.

Built-in Element Kinds

Identify the type of element.

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

Constants

`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.

Discussion
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.

Version Notes
Introduced with InputSprocket 1.0.

Device Emulation Flag

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
};
```

Constants

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

Discussion

Version Notes
Introduced with InputSprocket

Element Label Constants

Indicate preferred usage for the various elements.

```
enum {
    kISpElementLabel_None = 'none',
    kISpElementLabel_Axis_XAxis = 'xaxi',
    kISpElementLabel_Axis_YAxis = 'yaxi',
    kISpElementLabel_Axis_ZAxis = 'zaxi',
    kISpElementLabel_Axis_Rx = 'rxax',
    kISpElementLabel_Axis_Ry = 'ryax',
    kISpElementLabel_Axis_Rz = 'rzax',
    kISpElementLabel_Axis_Roll = 'rzax',
    kISpElementLabel_Axis_Pitch = 'rxax',
    kISpElementLabel_Axis_Yaw = 'ryax',
    kISpElementLabel_Axis_RollTrim = 'rxtm',
    kISpElementLabel_Axis_PitchTrim = 'trim',
    kISpElementLabel_Axis_YawTrim = 'rytm',
    kISpElementLabel_Axis_Gas = 'gasp',
    kISpElementLabel_Axis_Brake = 'brak',
    kISpElementLabel_Axis_Clutch = 'cltc',
    kISpElementLabel_Axis_Throttle = 'thrt',
    kISpElementLabel_Axis_Trim = 'trim',
    kISpElementLabel_Axis_Rudder = 'rudd',
    kISpElementLabel_Axis_ToeBrake = 'toeb',
    kISpElementLabel_Delta_X = 'xdlt',
    kISpElementLabel_Delta_Y = 'ydlt',
    kISpElementLabel_Delta_Z = 'zdlt',
    kISpElementLabel_Delta_Rx = 'rxdl',
    kISpElementLabel_Delta_Ry = 'rydl',
    kISpElementLabel_Delta_Rz = 'rzdl',
    kISpElementLabel_Delta_Roll = 'rzdl',
    kISpElementLabel_Delta_Pitch = 'rxdl',
    kISpElementLabel_Delta_Yaw = 'rydl',
    kISpElementLabel_Delta_Cursor_X = 'curx',
    kISpElementLabel_Delta_Cursor_Y = 'cury',
    kISpElementLabel_Pad_POV = 'povh',
    kISpElementLabel_Pad_Move = 'move',
    kISpElementLabel_Pad_POV_Horiz = 'hpov',
    kISpElementLabel_Pad_Move_Horiz = 'hmov',
    kISpElementLabel_Btn_Fire = 'fire',
    kISpElementLabel_Btn_SecondaryFire = 'sfir',
    kISpElementLabel_Btn_Jump = 'jump',
    kISpElementLabel_Btn_Quit = 'strt',
    kISpElementLabel_Btn_StartPause = 'paus',
    kISpElementLabel_Btn_Select = 'optn',
    kISpElementLabel_Btn_SlideLeft = 'blft',
    kISpElementLabel_Btn_SlideRight = 'brgt',
    kISpElementLabel_Btn_MoveForward = 'btmf',
    kISpElementLabel_Btn_MoveBackward = 'btmb',
    kISpElementLabel_Btn_TurnLeft = 'bttl',
    kISpElementLabel_Btn_TurnRight = 'bttr',
    kISpElementLabel_Btn_LookLeft = 'btll',
    kISpElementLabel_Btn_LookRight = 'btlr',
    kISpElementLabel_Btn_LookUp = 'btlu',
    kISpElementLabel_Btn_LookDown = 'btld',
    kISpElementLabel_Btn_Next = 'btnx',
    kISpElementLabel_Btn_Previous = 'btpv',
    kISpElementLabel_Btn_SideStep = 'side',
    kISpElementLabel_Btn_Run = 'quik',
    kISpElementLabel_Btn_Look = 'blok',
    kISpElementLabel_Btn_Minimum = 'min ',
```

```
    kISpElementLabel_Btn_Decrement = 'decr',
    kISpElementLabel_Btn_Center = 'cent',
    kISpElementLabel_Btn_Increment = 'incr',
    kISpElementLabel_Btn_Maximum = 'max ',
    kISpElementLabel_Btn_10Percent = ' 10 ',
    kISpElementLabel_Btn_20Percent = ' 20 ',
    kISpElementLabel_Btn_30Percent = ' 30 ',
    kISpElementLabel_Btn_40Percent = ' 40 ',
    kISpElementLabel_Btn_50Percent = ' 50 ',
    kISpElementLabel_Btn_60Percent = ' 60 ',
    kISpElementLabel_Btn_70Percent = ' 70 ',
    kISpElementLabel_Btn_80Percent = ' 80 ',
    kISpElementLabel_Btn_90Percent = ' 90 ',
    kISpElementLabel_Btn_MouseOne = 'mou1',
    kISpElementLabel_Btn_MouseTwo = 'mou2',
    kISpElementLabel_Btn_MouseThree = 'mou3'
};
```

Constants

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

`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.

`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.

`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).

`kISpElementLabel_Btn_Fire`
The element should be a fire button.

`kISpElementLabel_Btn_SecondaryFire`
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_LookLeft`
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.

Discussion
InputSprocket uses element label constants to help autoconfigure devices and to determine which labels appear as valid choices for certain controls. You can use these constants in a need structure (that is, a structure of type `ISpElementInfo`) to indicate preferred usage for each type of element:

Version Notes
These element-based labels introduced with InputSprocket 1.0.2.

Element List Flag

Specifies allocating an element list in temporary memory.

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

Constants

`kISpElementListFlag_UseTempMem`
Allocate the element list in temporary memory.

Discussion

Use when calling the `ISpElementList_New` function.

Version Notes

Introduced with InputSprocket 1.0.

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
};
```

Constants

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

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

`kISpNoneIconSuite`
Icon to indicate that no need currently maps to that element.

kISpElementLabel_XAxis

```
enum {
    kISpElementLabel_XAxis = 'xaxi',
    kISpElementLabel_YAxis = 'yaxi',
    kISpElementLabel_ZAxis = 'zaxi',
    kISpElementLabel_Rx = 'rxax',
    kISpElementLabel_Ry = 'ryax',
    kISpElementLabel_Rz = 'rzax',
    kISpElementLabel_Gas = 'gasp',
    kISpElementLabel_Brake = 'brak',
    kISpElementLabel_Clutch = 'cltc',
    kISpElementLabel_Throttle = 'thrt',
    kISpElementLabel_Trim = 'trim',
    kISpElementLabel_POVHat = 'povh',
    kISpElementLabel_PadMove = 'move',
    kISpElementLabel_Fire = 'fire',
    kISpElementLabel_Start = 'strt',
    kISpElementLabel_Select = 'optn',
    kISpElementLabel_Btn_PauseResume = 'strt'
};
```

Constants

kISpElementLabel_XAxis
kISpElementLabel_YAxis
kISpElementLabel_ZAxis
kISpElementLabel_Rx
kISpElementLabel_Ry
kISpElementLabel_Rz
kISpElementLabel_Gas
kISpElementLabel_Brake
kISpElementLabel_Clutch
kISpElementLabel_Throttle
kISpElementLabel_Trim
kISpElementLabel_POVHat
kISpElementLabel_PadMove
kISpElementLabel_Fire
kISpElementLabel_Start
kISpElementLabel_Select
kISpElementLabel_Btn_PauseResume

Discussion

Version Notes

kISpIconTransform_Selected

```
enum {
    kISpIconTransform_Selected = 16384,
    kISpIconTransform_PlotIcon = 256,
    kISpIconTransform_PlotPopupButton = 768,
    kISpIconTransform_PlotButton = 1024,
    kISpIconTransform_DeviceActive = 3
};
```

Constants

```
kISpIconTransform_Selected
kISpIconTransform_PlotIcon
kISpIconTransform_PlotPopupButton
kISpIconTransform_PlotButton
kISpIconTransform_DeviceActive
```

Discussion

Version Notes

kISpNeedFlag_NoMultiConfig

```
enum {
    kISpNeedFlag_NoMultiConfig = 1,
    kISpNeedFlag_Utility = 2,
    kISpNeedFlag_PolledOnly = 4,
    kISpNeedFlag_EventsOnly = 8,
    kISpNeedFlag_NoAutoConfig = 16,
    kISpNeedFlag_NoConfig = 32,
    kISpNeedFlag_Invisible = 64,
    kISpNeedFlag_Button_AlreadyAxis = 268435456,
    kISpNeedFlag_Button_ClickToggles = 536870912,
    kISpNeedFlag_Button_ActiveWhenDown = 1073741824,
    kISpNeedFlag_Button_AlreadyDelta = -2147483648,
    kISpNeedFlag_Axis_AlreadyButton = 268435456,
    kISpNeedFlag_Axis_Asymetric = 536870912,
    kISpNeedFlag_Axis_AlreadyDelta = 1073741824,
    kISpNeedFlag_Delta_AlreadyAxis = 268435456,
    kISpNeedFlag_Delta_AlreadyButton = 536870912
};
```

Constants

`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.

Discussion

Version Notes

kOSType_ISpDriverFileType

```
enum {
    kOSType_ISpDriverFileType = 'shlb',
    kOSType_ISpDriverCreator = 'insp'
};
```

Constants

`kOSType_ISpDriverFileType`

`kOSType_ISpDriverCreator`

Discussion

Version Notes

kSSpMedium_Air

```
enum {
    kSSpMedium_Air = 0,
    kSSpMedium_Water = 1
};
```

Constants

`kSSpMedium_Air`

`kSSpMedium_Water`

Discussion

Version Notes

kSSpSourceMode_Unfiltered

```
enum {
    kSSpSourceMode_Unfiltered = 0,
    kSSpSourceMode_Localized = 1,
    kSSpSourceMode_Ambient = 2,
    kSSpSourceMode_Binaural = 3
};
```

Constants

`kSSpSourceMode_Unfiltered`

`kSSpSourceMode_Localized`

`kSSpSourceMode_Ambient`

`kSSpSourceMode_Binaural`

Discussion

Version Notes

kSSpSpeakerKind_Stereo

```
enum {
    kSSpSpeakerKind_Stereo = 0,
    kSSpSpeakerKind_Mono = 1,
    kSSpSpeakerKind_Headphones = 2
};
```

Constants

`kSSpSpeakerKind_Stereo`

`kSSpSpeakerKind_Mono`

`kSSpSpeakerKind_Headphones`

Discussion

Version Notes

Maximum String Length Constants

Indicate the maximum allowable length for various strings.

```
enum {
    kNSpMaxPlayerNameLen = 31,
    kNSpMaxGroupNameLen = 31,
    kNSpMaxPasswordLen = 31,
    kNSpMaxGameNameLen = 31,
    kNSpMaxDefinitionStringLen = 255
};
```

Constants

`kNSpMaxPlayerNameLen`
: The maximum length for a player's name.

`kNSpMaxGroupNameLen`
: The maximum length for a group.

`kNSpMaxPasswordLen`
: The maximum length for a password (used to join a game).

`kNSpMaxGameNameLen`
: The maximum length for the name of the game.

`kNSpMaxDefinitionStringLen`
: The maximum allowable string length.

Discussion

These constants define the maximum lengths allowed for various strings used in NetSprocket. You should use these constants in place of hardcoded values when checking for string lengths.

Version Notes

Introduced with NetSprocket 1.0.

Network Message Delivery Flags

Specify delivery instructions for network messages.

```
enum {
    kNSpSendFlag_FailIfPipeFull = 1,
    kNSpSendFlag_SelfSend = 2,
    kNSpSendFlag_Blocking = 4
};
```

Constants

`kNSpSendFlag_FailIfPipeFull`
: NetSprocket will not accept the network message you are attempting to send if there are too many messages pending in the output buffer. Use this if you want to send data that is extremely time critical and useless if not delivered immediately.

`kNSpSendFlag_SelfSend`
: This flag is used to instruct NetSprocket to send a copy of this message to yourself as a player in addition to any other players or groups it is addressed to. You will receive a copy of your message in the message queue. If you send a message to all players (`kNSpAllPlayers`) without setting this flag, NetSprocket will not deliver the message to the sender.

`kNSpSendFlag_Blocking`
: This flag is used to have NetSprocket block the call and not return until the message has been successfully sent. The combination of `kNSpSendFlag_Blocking` and `kNSpRegistered` may cause your application to wait a significant period of time before satisfying these requirements, because it will wait until all the recipients have acknowledged receipt of the message or the retry limit has been reached.

Discussion

These constants are message delivery flags to assist you in determining and controlling the status of message delivery. You can OR these constants together with the network message priority flags.

A message that is successfully sent does not ensure receipt by the intended players unless `kNSpRegistered` is specified. It simply means that NetSprocket successfully delivered the message to the appropriate network protocol handler and the message has been duly passed on.

In NetSprocket version 1.0, a message sent from any player who is not the host with this flag set will return when the message has been delivered to the host. The message may or may not have been received by all of the intended recipients.

Version Notes

Introduced with NetSprocket 1.0.

Network Message Priority Flags

Indicate delivery priorities for network messages.

```
enum {
    kNSpSendFlag_Junk = 1048576,
    kNSpSendFlag_Normal = 2097152,
    kNSpSendFlag_Registered = 4194304
};
```

Constants

`kNSpSendFlag_Junk`
: This message is junk mail. This type of message will be sent only when no other messages of higher priority are pending. This is essentially a “fire and forget” message. Delivery will only be attempted once, and there is no guarantee of receipt.

`kNSpSendFlag_Normal`
: This message is an ordinary, every-day message. It will be sent immediately, but like `kNSpSendFlag_Junk`, delivery will only be attempted once, and there is no guarantee of receipt.

`kNSpSendFlag_Registered`
: Like registered mail, this message is quite important. Delivery is of the highest priority. For example, if `kNSpSendFlag_Normal` or `kNSpSendFlag_Junk` messages are being sent (or if a message is being chunked for delivery in multiple packets), they will be interrupted in favor of a `kNSpRegistered` message. NetSprocket will demand proof of receipt and will continue retrying until the maximum retry limit has been exceeded.

Discussion
These constants are used to identify various priorities you may assign to network messages using a mail service metaphor. You use these flags in the `NSpFlags` parameter of the function `NSpMessage_Send` (page 92).

Version Notes
Introduced with NetSprocket 1.0.

Network Message Type Constants

Indicate various message types.

```
enum {
    kNSpSystemMessagePrefix = -2147483648,
    kNSpError = -1,
    kNSpJoinRequest = -2147483647,
    kNSpJoinApproved = -2147483646,
    kNSpJoinDenied = -2147483645,
    kNSpPlayerJoined = -2147483644,
    kNSpPlayerLeft = -2147483643,
    kNSpHostChanged = -2147483642,
    kNSpGameTerminated = -2147483641,
    kNSpGroupCreated = -2147483640,
    kNSpGroupDeleted = -2147483639,
    kNSpPlayerAddedToGroup = -2147483638,
    kNSpPlayerRemovedFromGroup = -2147483637,
    kNSpPlayerTypeChanged = -2147483636
};
```

Constants
`kNSpSystemMessagePrefix`
: This is the prefix of all NetSprocket system messages. You can OR a message's `what` field with this constant to determine if the message is a system message.

`kNSpError`
: A local error has occurred. It may have occurred when receiving a message, attempting to send a message, or attempting to allocate memory.

`kNSpJoinRequest`
: A player wants to join a game. You do not need to respond to this message. NetSprocket will either use the default password check, or your custom join handler (if installed) to approve or deny the join request.

`kNSpJoinApproved`
: Your request to join a game has been approved.

`kNSpJoinDenied`
Your request to join a game has been denied.

`kNSpPlayerJoined`
A player has joined the game.

`kNSpPlayerLeft`
A player has left the game.

`kNSpHostChanged`
The host of the game has changed. This message type is unused as NetSprocket does not currently support host renegotiation.

`kNSpGameTerminated`
The game has been permanently stopped.

`kNSpGroupCreated`
Someone has created a group.

`kNSpGroupDeleted`
Someone has deleted a group.

`kNSpPlayerAddedToGroup`
A player was added to a group.

`kNSpPlayerRemovedFromGroup`
A player was removed from a group.

`kNSpPlayerTypeChanged`
A player's type was changed.

Discussion
These constants are used to identify standard message types when passed in a message header. NetSprocket uses these types to clearly identify the network messages so you can process the message with the appropriate data structure.

All message types with negative values are reserved for use by Apple Computer, Inc.

Version Notes
Introduced with NetSprocket 1.0.

Options for Hosting, Joining, and Disposing Games

Specify various game options.

```
enum {
    kNSpGameFlag_DontAdvertise = 1,
    kNSpGameFlag_ForceTerminateGame = 2
};
```

Constants
`kNSpGameFlag_DontAdvertise`
When this flag is passed with `NSpGame_Host`, the game object is created, but the game is not advertised on any protocols. By default, a call to `NSpGame_Host` advertises the game on the protocols in the protocol list.

`kNSpGameFlag_ForceTerminateGame`
When the host calls `NSpGame_Delete` with this flag set, NetSprocket will end the game without attempting to find a host replacement. All the players will receive a message that the game has been ended, and any further calls from them will return an error. Normally, a call to `NSpGame_Delete` by the host will cause NetSprocket to negotiate a new host.

Discussion
These constants are used to control games. You use these constants in the `inFlags` parameter of the `NSpGame_Host` (page 80) , `NSpGame_Join` (page 82) , and `NSpGame_Dispose` (page 78) functions.

Version Notes
Introduced with NetSprocket 1.0.

Reserved Player IDs

Indicate special recipients when sending network messages.

```
enum {
    kNSpAllPlayers = 0,
    kNSpHostOnly = -1
};
```

Constants
`kNSpAllPlayers`
Send the message to all players.

`kNSpHostOnly`
Send the message to the player currently hosting the game.

Discussion
These constants are used to identify player IDs that are reserved for message delivery. Specify one of these special IDs in the `to` field of a message structure.

It is possible for the host to change during the course of a game. It is also possible for a host to not have a player ID, because someone may host a game without participating as a player. Therefore you should not use a player ID to send a message to the host. Instead, you should use `kNSpHostOnly` reserved for a host.

Version Notes
Introduced with NetSprocket 1.0.

Resource Types

These constants identify resource types defined by InputSprocket. See “Resources”.

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

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

`kISpSetListResourceType`
A resource of type 'setl'.

`kISpSetDataResourceType`
A resource of type 'tset'.

Discussion

Version Notes
Introduced with InputSprocket 1.0.

Topology Types

Indicate network topology types.

```
enum {
    kNSpClientServer = 1
};
```

Constants
`kNSpClientServer`
Client/server topology.

Discussion
You use these constants to identify the topology you are choosing for your game. You pass this value in the `inTopology` field of `NSpGame_Host` (page 80).

NetSprocket version 1.0 currently supports only client/server topology.

Version Notes
Introduced with NetSprocket 1.0.

Virtual Element Flag

Specifies allocating virtual elements in temporary memory.

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

Constants
`kISpVirtualElementFlag_UseTempMem`
Allocate the virtual element in temporary memory.

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

Version Notes
Introduced with InputSprocket 1.0.

Document Revision History

This table describes the changes to *Apple Game Sprockets Legacy Reference*.

Date	Notes
2003-01-01	New document extracted from the original Apple Game Sprockets Reference.

Document Revision History

Index

K

M

N

O

R

S

T

V