
# IEEE 1394 SDK

## Version 1.0.2

www.radisys.com

World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044

International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620

RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Revision B
August 2001

## Copyright and publication information

This manual reflects version 1.0.2 of IEEE 1394 SDK. Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

## Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

## Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

August 2001

# Table of Contents

# Chapter 1: Introduction

This chapter provides an overview of the IEEE 1394 Software Developer's Kit (SDK). It includes the following sections:

**Items Included in This Package**

**IEEE 1394 Overview**

# Items Included in This Package

IEEE 1394 SDK contains the following software and documentation.

## Software

The following software is included in this package:

- IEEE 1394 SDK

## Documentation

The documentation set for IEEE 1394 SDK includes the following:

- documentation for all OS-9 components included on the IEEE 1394 SDK CD-ROM (PDF format)
- release notes providing late-breaking information that could not included in the manuals (PDF format)
- this ***IEEE 1394 SDK*** manual

# IEEE 1394 Overview

The IEEE 1394 High Performance Serial Bus is an IEEE standard for connecting devices like camcorders, set-top boxes, and scanners. The goal of the IEEE 1394 protocol is to provide easy-to-use, high-speed communications.

The IEEE 1394 standard is comprised of a single plug-and-socket connection on which up to 63 devices can be attached with data transfer speeds up to 400 MBPS. In addition, its implementations provide the following:

- a thin serial cable rather than a thicker parallel cable
- a plug-in serial connector on the back of the host system and many peripheral devices
- hot-plug and plug-and-play capability that does not disrupt the host system
- peer-to-peer operation independent of host system architecture
- the ability to chain devices together in different ways without requiring terminators or complicated set-up requirements

## Performance

There are two levels of interface in the IEEE 1394, including one for the backplane bus within the computer and another for the point-to-point interface between the device and the computer on the serial cable; a simple bridge connects the two environments.

The backplane bus supports 12.5, 25, or 50 megabits per second of data transfer. The cable interface supports 100, 200, or 400 megabits per second. Each of these interfaces can handle any of the possible data rates and change from one to the other, as needed.

The serial bus functions as though devices are in slots within the computer, sharing a common memory space. A 64-bit device address allows flexibility in configuring devices in chains and trees from a single socket.

In addition, IEEE 1394 supports two types of data transfer, including asynchronous and isochronous. Asynchronous transfer is appropriate for traditional applications, while isochronous is useful for multimedia applications because it provides transport at a pre-determined rate.

## The Protocol Stack

IEEE 1394 is comprised of four protocol layers. Each layer has an associated set of services designed to support communication between application and layer. The four layers include the following:

- Bus Management

  This layer connects a wide range of devices that do not require a PC or other bus controller. It involves the following services:

  > a cycle master that broadcasts cycle start packets (required for isochronous operation)

  > an isochronous resource manager, if any nodes support isochronous communication

  > an optional bus master

- Transaction

  This layer supports the Control and Status Register (CSR) architecture request-response protocols for read, write, and lock operations related to asynchronous transfers.

- Link

  This layer provides the translation of a transaction layer request or response into a corresponding packet or sub-action delivered over the serial bus. In addition, Cyclic Redundancy Checking (CRC) is performed in the link layer.

  The link layer also provides address and channel number decoding for incoming asynchronous or isochronous packets. Along with this, it provides a means for accessing the PHY registers, as defined in Annex J of the IEEE 1394 specification via the PHY control register.

- Physical

  This layer provides an electrical and mechanical interface necessary for transmission and reception of data bits transferred across the serial bus. In addition, this layer provides the initialization and arbitration services necessary to insure that only one node is sending data at a time. It also translates the serial bus data stream and signal levels into those required by the link layer. The isolation may be implemented between the physical layer and the link layer. In isolation, the bus conductors power the chip that implements the physical layer.

  On bus reset, the structure of the bus is determined, the node IDs (physical addresses) are assigned to each node, and the arbitration for the cycle master, the isochronous resource manager, and the bus master node occur. Any resources not reclaimed become available for future use. After this delay, new resources may be allocated.

## Communication

Both asynchronous and isochronous communication are addressed in the link layer of IEEE 1394. The link layer provides address and channel number decoding for incoming asynchronous and isochronous packets.

### Asynchronous

Due to the nature of the 1394 specification, only node specific data and broadcast data are received from the link layer controller. From the link layer driver interrupt service routine, the SoftStax `spf_rx` thread is called to move asynchronous data up the stack to `sp1394`. The asynchronous data is de-multiplexed at `sp1394` based on the transaction label corresponding to the appropriate application/fireman or the CSR map.

Applications mapping IEEE 1212 address space for their own use map the address space with the sp1394 module by means of a setstat (ioctl). They may register to be called (via a “pre-callback” routine) upon receiving transactions directed at a mapped address, or instruct sp1394 to perform necessary tasks on their behalf and make a post receive callback notification to inform the application of the action taken.

## Isochronous

Isochronous data is routed directly from the link layer based on the channel number to the isochronous application. Applications using isochronous data transmission or reception make calls that translate into setstats (ioctl) to sp1394. The module provides routines to allocate and free resources such as channel and bandwidth. In addition, it makes calls to attach appropriate buffers to the path and channel. The link layer driver uses these buffers to either transmit or receive data. Received data is moved into the buffer directly in the interrupt service routine.

Furthermore, the hardware may be instructed to DMA directly into this buffer, thus providing for a zero-copy environment. In the event that the data is fed immediately into another hardware chip, such as an MPEG decoder, the data can be directly dropped into the MPEG decoder buffers and the driver of the chip informed of the arrival of data. This provides for tremendous flexibility for the OEM. No data is moved up the stack; the application is notified when its buffers contains the desired data. On reception, the application may also choose to be notified when a high watermark is reached. By avoiding data movement through the spf_rx thread, you also avoid delays of context switches.

# Chapter 2: Usage and Functionality

This chapter provides the usage and functionality for the IEEE 1394 Software Developer's Kit (SDK) applications. It includes the following sections:

- **Requirements and Compatibility**
- **Getting Started**
- **Utilities**
- **Application Programs**

# Requirements and Compatibility

Below are the software and hardware requirements for IEEE 1394.

## Software Requirements

Your reference board should have the following loaded:

- SPF/*SoftStax* Environment (Edition 264 of the SPF File Manager)
- sysmbuf (the OS-9 communication buffer facility)

  Use `mbinstall` to install sysmbuf before initializing the 1394 stack.
- debug library (`dbg_mod.l`) for making the debug version of the link layer driver.

## Hardware Requirements and Settings

The IEEE 1394 SDK requires the following hardware:

- SH3 (SH7709 or SH7709A) or SH4 (SH7750) reference board
- Hitachi IEEE daughter board MS1394DB01

### Settings

- S1: set switch 2 to OFF. Set all other switches to ON.
- S2: set all switches to OFF.
- S3 (`IRQ select`): set switch 2 to ON. Set all other switches to OFF.
- S4 (`CS select`): set switch 1 to ON. Set all other switches to OFF.
- J1: no jumpers
- JP1: jumper present
- JP2: no jumper
- JP3: no jumper

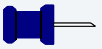
# Getting Started

Complete the following steps to begin working with IEEE 1394:

Step 1. Install the sysmbuf using `mbinstall`.

Step 2. Run `fireman` in the background; `fireman` is responsible for starting the stack and enabling the serial bus management (if configured through the `lustat`).

Step 3. Run the IEEE 1394 application.



**Note**

If you are testing your system for connectivity, the `1394info` utility may be used to show information for all devices on the serial bus.

# Utilities

Currently, only one utility is included for the IEEE 1394 SDK. This utility, `1394info`, is described below.

# 1394info

Show devices on IEEE1394 bus

## Syntax

```
1394info [<opts>]
```

## OS

OS-9 for 68K; OS-9

## Description

The `1394info` utility is used to show all devices on the IEEE1394 bus. In non-verbose mode, it displays one node per line, with an asterisk prefixing the current node. In verbose mode, additional information such as `Self Info`, `Bus Info`, `Topology Map`, and `Speed Map` are also displayed.

## Note

Before using this utility, the IEEE1394 stack must be initialized. The following commands should be executed before running `1394info` for the first time since booting:

- `mbinstall`
- `fireman &`

## Options

| | |
|---|---|
| `-v` | display verbose information, including the Control and Status Registers (CSR) and ROM directory space |
| `-vv` | display more information, including hexadecimal display information |

## Example Output

Running a simple `1394info` should provide output similar to the following:

```
$ 1394info
Index  DevType   Serial    NodeID
  0    08002851  00001643  FFC0
* 1    01234567  89ABCDEF  FFC1
```

In the case above, the host on which the command was typed is not the bus master. The bus master is always at `FFC0` on a IEEE1394 serial bus.

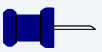
# Application Programs

The following section describes the uses of the IEEE 1394 example application programs.

## Implementation of a Signal Handler

All applications must implement a signal handler; applications are informed of bus resets via two signals (`SIG_1394BUSRESET` and `SIG_1394BUSRESET_DONE`). They should be capable of handling these signals in their signal handlers.

On receipt of the signal, the application chooses the appropriate action to take. The action may include unmapping all current addresses, as well as terminating or re-registering with the protocol stack.



### Note

Currently, on a bus reset the protocol stack does not unmap the address range that the application may have mapped.

### Example

```
extern void *_glob_data;  /* for _os_intercept */
/*************************************************************
 *  sighand - handle signal

*************************************************************/
void sighand(signal_code sig) {
  switch (sig) {
    case SIG_1394BUSRESET :
     /* Take appropriate action */
      break;
  case SIG_1394BUSRESET_DONE :
     /* Take appropriate action */
      break;
  _os_rte();
}
```

```
main() {
  extern void *_glob_data;
   /* Install the intercept handler */
   _os_intercept( sighand, _glob_data);
   /* Do other things */
}
```

# Applications

The section describes additional example applications used with IEEE 1394.

## test1394

The following provides the command line syntax for `test1394`:

```
<GUID: DevType> <GUID: Serial>
```

The `test1394` application performs the following:

Step 1. installs the signal handler

Step 2. initializes the stack (`mw1394Initialize`)

Step 3. calls `mw1394GetNodeIDbByGUID` to obtain the node id of the node with a global unique identifier provided on the command line

This identifier can be acquired using the `1394info` utility.

Step 4. maps a sample 1212 address range (`0xFF0000000000`)

Step 5. illustrates a callback function template (`mw1394MapAddressRange`)

Step 6. makes a call to get the topology map (`mw1394GetTopologyMap`)

Step 7. calls `mw1394AsynchWrite` to write a data quadlet to a remote node with mapped address `0xFF0F00000000`

Step 8. calls `mw1394AsynchRead` to read the CSR ROM location of a remote node as data quadlets

Step 9. makes a single call to `mw1394AsynchRead` to read the CSR ROM location of a remote node as a data block

Step 10. makes a call to `mwGetBusInfoBlock` to obtain the bus information block of the destination

Step 11. sends a lock request to the remote node (`mw1394AsynchLock`)

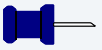
Step 12. issues a bus reset (`mw1394ResetBus`)

Step 13. unmaps the address range (`mw1394UnMapAddressRange`)

Step 14. terminates the stack (`mw1394Terminate`)

## isoxmit and isorecv

Designed to work in tandem, these applications can isochronously transfer a file on one node to the other, each running OS-9.



### Note

Some isochronous data may be lost; therefore, the entire file might not be received by `isorecv`.

Start the `isorecv` program on one node before starting the `isoxmit` program on the other.

### isoxmit

`isoxmit` is the program that isochronously transmits a file on the node. The following provides the command line syntax for `isoxmit`:

```
isoxmit <GUID:DevType> <GUID:Serial> <filename>
```

`GUID` is the global unique identifier of the node receiving the data; `filename` is the name of the file to be transmitted isochronously.

The `isoxmit` application performs the following:

| | |
|---|---|
| Step 1. | installs the signal handler |
| Step 2. | Initializes the stack and opens a connection |
| Step 3. | maps the address `0xFFFF00000000` to the `recvReady` variable |
| Step 4. | opens the specified file for reading and determines the size of the file |
| Step 5. | calls `mw1394GetNodeIDByGUID` to obtain the node id of the receiver |
| Step 6. | calls `mw1394AsynchWrite` to write the file size to a mapped address `0xFFFF00000000` on the receiver |
| Step 7. | calls `mw1394AsynchWrite` to a mapped address `0xFFF000000000` on the receiver, thereby notifying the receiver that it is done writing the file size |
| Step 8. | allocates sufficient memory using `_os_srqmem` for `buffer` and reads the data out of the file into the buffer |
| Step 9. | closes the file |
| Step 10. | waits on the `recvReady` flag to be set by the receiver to indicate that the receiver is ready to accept isochronous data |
| Step 11. | once the `recvReady` flag is set, it calls `mw1394UnMapAddressRange` to unmap the previously mapped address range |
| Step 12. | calls `mw1394IsochAllocateChannel` to request the isochronous bus manager for a specific channel (34) |
| Step 13. | calls `mw1394IsochAllocateBandwidth` to request the isochronous bus manager for bandwidth to transmit 488 bytes per frame (at the application level) |
| Step 14. | calls `mw1394IsochAttachBuffers` to attach the buffers with the specified path and channel |
| Step 15. | calls `mw1394IsochXmit` to transmit the data on the allocated channel |
| Step 16. | calls `mw1394IsochDetachBuffers` to detach the buffers previously allocated |

Step 17. calls `mw1394IsochFreeBandwidth` to free the previously allocated bandwidth

Step 18. calls `mw1394IsochFreeChannel` to free the previously allocated channel

Step 19. calls `mw1394Terminate` to close the connection

Step 20. calls `_os_srtmem` to free the memory buffer allocated before exiting

---

### isorecv

`isorecv` is the program that receives the data transmitted isochronously. The following provides the command line syntax for `isorecv`:

```
isorecv  <GUID: DevType> <GUID: Serial> <filename>
```

`GUID` is the global unique identifier of the node transmitting the data; `filename` is the name of the file to save the data received.

The `isorecv` application performs the following:

---

Step 1. installs the signal handler

Step 2. initializes the stack and opens a connection

Step 3. calls `mw1394GetNodeIDByGUID` to obtain the node id of the sender

Step 4. maps the address `0xFFFF00000000` to the `actual_fsize` variable

Step 5. maps the address `0xFFF000000000` to the `size_known` variable

Step 6. waits on the `size_known` flag to be set by the transmitter indicating that the transmitter has written the file size into the mapped address

Step 7. allocates sufficient memory space (by calling `_os_srqmem`) to enter in the isochronous data

Step 8. calls `mw1394IsochAttachBuffers` to attach the buffer, allocated above, with the specified path and channel

Step 9. calls `mw1394IsochListen` to start listening for data on the allocated channel

Step 10. notifies the transmitter that it is ready to receive the isochronous data by calling `mw1394AsynchWrite` to the mapped address `0xFFFF00000000` on the transmitter.

Step 11. When the data has arrived, the application issues a `STOP` request to stop reception and activity on this channel (by calling `mw1394IsochStop`).

Step 12. calls `mw1394IsochDetachBuffers` to detach the previously allocated buffers

Step 13. calls `mw1394UnMapAddressRange` to unmap the previously mapped address ranges

Step 14. calls `mw1394Terminate` to close the connection

Step 15. The data collected is written to the file specified on the command line. If no isochronous data packets are missed, the files should match (use `cmp` to test this) both on the transmitting node and on the receiving node.

Step 16. The memory buffer allocated is freed before returning.

## prenotify and prenotifyd

`prenotify` and `prenotifyd` show the pre-notification callback routines that the programmer can use to tell the stack to notify it when an asynchronous operation is requested in the mapped address range. This is performed by calling a callback function. The `prenotifyd` daemon initializes the stack and maps the address range starting at `FF0000000000` for a length of 80 bytes. The pre-notify application is the peer program that causes the callback routine to be called by making calls similar to those in **test1394**.

**Note**

Caution should be taken when using the above callback feature. The data is sent up in the system state context in mbufs. The callback should not attempt to free these mbufs. It may perform the necessary operation and return the appropriate `RCODE`. Because the call is from system context, sleeping in this routine brings the system to a halt.

## Configuring the Link Layer Driver Logical Unit Statics

The following fields of the link layer driver logical descriptors are configurable in the device descriptor. This can be done by defining the appropriate macro in the `spf_desc.h` file in the port directory.

The following fields are defined in the `SPF_LUSTAT` macro in `defs.h`:

| | |
|---|---|
| `lu_dbg_name` | the name of the debug module name |
| `lu_priority` | the interrupt polling priority |
| `lu_vector` | the interrupt vector number |
| `lu_io_base` | the port address on the board |
| `lu_SplitTimeLimit` | the split timeout limit (not supported in `sp8412`) |
| `lu_SplitTimeOutEn` | the split timeout enable flag (not supported in `sp8412`) |
| `lu_AsySize` | the size of asynchronous buffer<br>The remaining part of the FIFO is used for isochronous operations. |
| `lu_ARBSize` | the size of the asynchronous receive buffer<br>The remaining part of the asynchronous FIFO is used for asynchronous transmit buffer. |

| | |
|---|---|
| `lu_IRBSize` | the size of the isochronous receive buffer<br><br>The remaining part of the isochronous receive buffer is used for the isochronous transmit buffer. |
| `myConfigROM` | the config ROM specification for the node<br><br>To modify this, change its value in `SPF_LUSTAT_INIT` in `defs.h` and rebuild the descriptor. |

# Chapter 3: API Functions

This chapter discusses API functions in IEEE 1394. The following sections are included:

- **API Functions**
- **Transfer Sequence**

# API Functions

All applications are currently implemented as user state programs that call into the *SoftStax* environment through the IEEE 1394 library. The IEEE 1394 provides the functions to perform all necessary asynchronous and isochronous operations through a common interface. Applications are informed about node and bus events by sending appropriate signals. For this reason, a signal handler is required.

## For More Information

For more information on signal handlers, see **Chapter 2: Usage and Functionality**.

The following API functions are currently supported in `lib1394`.

# mw1394Initialize()

Initialize Stack and Open New Path

## Syntax

```
#include <lib1394.h>
error_code mw1394Initialize(char *device, path_id *path)
```

## Libraries

`lib1394.l`

## Description

This function initializes the stack and opens a new path to it. If the stack is already initialized, it returns a new path. The path should be used for all communication.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `*device` | points to the protocol stack<br>**Example**<br>`/spff0/sptr0` |
| `*path` | points to the location where the path would be returned |

## Error Values

all errors of SPF stack initialization

# mw1394Terminate()

Terminates Stack and Closes Path

## Syntax

```
#include <lib1394.h>
error_code mw1394Terminate(path_id path)
```

## Libraries

`lib1394.l`

## Description

This function terminates the 1394 stack and closes the path that was opened at initialization time.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |

## Error Values

all errors of SPF stack termination

# mw1394GetBusInfo()

Obtain Bus Information

## Syntax

```
#include <lib1394.h>
error_code mw1394GetBusInfo(path_id path,struct bus_info *binfo)
```

## Libraries

`lib1394.l`

## Description

The following information about the bus is returned:

- number of nodes on the bus
- self node ID
- root node ID
- isochronous resource manager node ID
- bus manager ID
- cycle master node ID

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*binfo` | pointer to the 1394 bus information parameter block |

## Data Structures

The following describes the 1394 bus information parameter block:

```
typedef struct bus_info {
  u_int32 numNodes;
  u_int16 selfNodeID;
  u_int16 rootID;
  u_int16 irmID;
  u_int16 busMgrID;
  u_int16 cycleMasterID;
} bus_info, *Bus_info;
```

## Error Values

EOS_MW1394_BUSRESET

# mw1394GetIRMNodeID()

Return Node Identifier

## Syntax

```
#include <lib1394.h>
error_code mw1394GetIRMNodeID(path_id path,u_int16 *irmid)
```

## Libraries

`lib1394.l`

## Description

The function returns the 16-bit node identifier of the node currently acting as the isochronous resource manager on the bus.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*irmid` | pointer to the 16-bit field that holds the node id of the isochronous resource manager (IRM) |

## Error Values

| | |
|---|---|
| `EOS_MW1394_NOIRM` | no isochronous resource manager was found (008:258) |

# mw1394GetBusInfoBlock()

Returns Bus Information

## Syntax

```
#include <lib1394.h>
error_code mw1394GetBusInfoBlock(path_id path, u_int16 nodeid,
                                 quadlet *block, u_int8 *blksize)
```

## Libraries

`lib1394.l`

## Description

The function returns the bus information block of the specified node and its size.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `nodeid` | node id of the node |
| `*block` | pointer to the memory block where the bus information block contents are stored<br><br>It is the caller's responsibility to see that sufficient memory is allocated to copy the entire bus information block. |
| `*blksize` | pointer to the location where the size of the retrieved bus information block is stored |

### Error Values

| | |
|---|---|
| `EOS_MW1394_FAILED` | an internal failure was detected (008:261) |
| `EOS_MW1394_TIMEDOUT` | timed out waiting for response (008:260) |
| `EOS_MW1394_BUSRESET` | a 1394 bus reset was detected (008:257) |

# mw1394GetDeviceInfo()

Retrieves Device Node Information

## Syntax

```
#include <lib1394.h>
error_code mw1394GetDeviceInfo(path_id, u_int16 nodeid,
                               mw1394_device_info_pb *devinfo)
```

## Libraries

`lib1394.l`

## Description

This function retrieves device node information for a specified device number on the IEEE 1394 bus. Devices are always numbered from zero in a contiguous fashion. When a device is connected to or removed from the IEEE 1394 bus, all device nodes are re-numbered. The current bus manager is always `nodeid zero`. Using `mw1394GetDeviceInfo` and a simple loop, a program can quickly examine and identify all devices attached to the bus.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `nodeid` | node to examine (numbered from `0`) |
| `devinfo` | pointer to the parameter block that receives the node information |

### Data Structures

The following describes the block that receives the node information parameter block:

```
typedef struct mw1394_device_info_pb {
  u_int32  VendID_DevType;
  u_int32  SerialNo;
  u_int16  NodeID;
  u_int8   Active;
  u_int8   Reserved;
  char    Vendor[64];
  char    Model[64];
{ mw1394_device_info_pb;
```

## Structure Members

The following describes members of the above parameter block data structure:

| | |
|---|---|
| `vendID_DevType` | vendor ID/device type as read from the Control and Status Registers (CSR) root directory |
| `SerialNo` | serial number as read from the CSR root directory |
| `NodeID` | bus nodeid of the device (parameter `nodeid\|0xFFC0`) |
| `Active` | a flag indicating whether or not the node is active<br><br>Whether or not the node is active is determined by the presence of a CSR root directory. |
| `Vendor` | optional string containing the Vendor ID |
| `Model` | optional string that contains the Model and hardware version |

### Error Values

| | |
|---|---|
| `EOS_MW1394_TIMEDOUT` | request timed out<br>The time-out is most likely due to a present device not responding to memory request (such as one that has no software driver initialized) |
| `EOS_MW1394_FAILED` | request failed<br>The cause of the failed request may be that the bus was reset or that there is no device present at this nodeid or higher. |

# mw1394GetSelfNodeInfo()

Return Host Link Information Node

## Syntax

```
#include <lib1394.h>
error_code mw1394GetSelfNodeInfo(path_id path,
                                 struct link_info *linfo)
```

## Libraries

`lib1394.l`

## Description

This function returns the link information of the host node.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameter

| | |
|---|---|
| `path` | path id of the path that was opened at init time |
| `*linfo` | a pointer to the link information parameter block |

## Data Structures

The following describes the link information parameter block:

```
typedef struct link_info {
  u_int8 nodeID;
  u_int16 busID;
  u_int8 gap_count;
  u_int8 link_active;
  u_int8 PHY_Speed;
  u_int8 PHY_Delay;
  u_int8 contender;
  u_int8 num_ports;
  u_int8 Initiated_reset;
} link_info, *Link_info;
```

## Error Values

The link layer driver implements this function and is responsible for any error codes. Current drivers available from Microware always provide the current link information and return SUCCESS. However, `link_active` and `Initiated_reset` are not updated.

# mw1394GetNodeIDByGUID()

Obtain Node ID

## Syntax

```
#include <lib1394.h>
error_code mw1394GetNodeIDByGUID(path_id, u_init16 *nodeid,
                        int32 *guid)
```

## Libraries

`lib1394.l`

## Description

This function is used to obtain the node ID of a node on the bus based on the global unique identifier.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameter

| | |
|---|---|
| `path` | path id of the path that was opened at init time |
| `*nodeid` | pointer to the location where the node id of the node is stored |
| `*guid` | pointer to the 64-bit global unique identifier |

## Error Values

| | |
|---|---|
| `EOS_MW1394_NNF` | the desired node was not found on the bus (008:259) |
| `EOS_MW1394_TIMEDOUT` | timed out waiting for response (008:260) |
| `EOS_MW1394_FAILED` | an internal failure was detected (008:261) |
| `EOS_MW1394_BUSRESET` | a 1394 bus reset was detected (008:257) |

# mw1394AsynchRead()

Perform Asynchronous Read

## Syntax

```
#include <lib1394.h>
error_code mw1394AsynchRead(path_id path,
                            mw1394_asynch_read_pb *readpb)
```

## Libraries

`lib1394.l`

## Description

This function performs an asynchronous read request to a node and receives data to the specified buffer.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*readpb` | pointer to the 1394 asynchronous read parameter block |

## Data Structures

The following describes the 1394 asynchronous read parameter block:

```
typedef struct mw1394_asynch_read_pb {
  u_int32 destid;
  mw1394_offset offset;
  u_int32 numBytes;
  void *buffer; /* Pointer to the data */
  u_int32 ackCode;
  u_int32 respCode;
} mw1394_asynch_read_pb;
```

### Structure Members

The following describes members of the 1394 asynchronous read parameter block data structure:

| | |
|---|---|
| `destid` | the 16-bit node ID of the destination |
| `offset` | the 48-bit IEEE 1212 address for the read request |
| `numBytes` | the number of bytes to read |
| `buffer` | a buffer of `numBytes` available bytes that receives the results of the read transaction |
| `ackCode` | field where the acknowledge code received for the read request is stored<br>This field is valid only if the response was not received and `EOS_MW1394_TIMEDOUT` error is returned. |
| `respCode` | field where the response code contained in the response packet is saved |

### Error Values

| | |
|---|---|
| `EOS_MW1394_TIMEDOUT` | timed out waiting for response (008:260) |
| `EOS_MW1394_FAILED` | an internal failure was detected (008:261) |
| `EOS_MW1394_BUSRESET` | a 1394 bus reset was detected (008:257) |

# mw1394AsynchWrite()

Performs Asynchronous Write

## Syntax

```
#include <lib1394.h>
error_code mw1394AsynchWrite(path_id path,
                             mw1394_asynch_write_pb *writepb)
```

## Libraries

`lib1394.l`

## Description

This function performs an asynchronous write of data to the specified node.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*writepb` | pointer to the 1394 asynchronous write parameter block |

## Data Structures

The following describes the 1394 asynchronous write parameter block:

```
typedef struct mw1394_asynch_write_pb {
  u_int32 destid;
  mw1394_offset offset;
  u_int32 numBytes;
  void  *buffer;/* Pointer to the data */
  u_int32 ackCode;
  u_int32 respCode;
} mw1394_asynch_write_pb;
```

### Structure Members

The following describes the members of the 1394 asynchronous write parameter block data structure:

| | |
|---|---|
| `destid` | the 16 bit node ID of the destination |
| `offset` | the 48 bit IEEE 1212 address for the write request |
| `numBytes` | the number of bytes to write |
| `buffer` | a buffer containing the data to be transmitted |
| `ackCode` | the field in which the acknowledge code received for the write request is stored<br><br>This field is valid only if the response was not received and `EOS_MW1394_TIMEDOUT` error is returned. |
| `respCode` | the field in which the response code contained in the response packet is saved |

### Error Values

| | |
|---|---|
| `EOS_MW1394_TIMEDOUT` | timed out waiting for response (008:260) |
| `EOS_MW1394_FAILED` | an internal failure was detected (008:261) |
| `EOS_MW1394_BUSRESET` | a 1394 bus reset was detected (008:257) |
| `EOS_MW1394_BADSIZE` | the length of packet specified is either greater than the maximum allowable payload or the allocated asynchronous transmit buffer size (008:262) |

# mw1394AsynchLock()

Perform Asynchronous Lock

## Syntax

```
#include <lib1394.h>
error_code mw1394AsynchLock(path_id path,
                            mw1394_asynch_lock_pb *lockpb)
```

## Libraries

`lib1394.l`

## Description

This function requests an asynchronous lock operation on the specified node.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*lockpb` | pointer to the 1394 asynchronous lock parameter block |

## Data Structures

The following describes the 1394 asynchronous lock parameter block:

```
typedef struct mw1394_asynch_lock_pb {
  u_int32 destid;
  mw1394_offset offset;
  u_int32 xtndTCode;
  u_int32 numArgBytes;
  u_int32 argValue[2];
  u_int32 numDataBytes;
  u_int32 dataValue[2];
  void    *buffer;
  u_int32 ackCode;
  u_int32 respCode;
} mw1394_asynch_lock_pb;
```

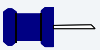
## Structure Members

The following describes members of the 1394 asynchronous lock parameter block data structure:

| | |
|---|---|
| `destid` | the 16-bit node ID of the destination |
| `offset` | the 48-bit IEEE 1212 address for the read request |
| `xtndTCode` | the extended transaction code |

**Possible Values for `xtndTCode`**

`XTCODE_MASK_SWAP`
`XTCODE_COMPARE_SWAP`
`XTCODE_FETCH_ADD`
`XTCODE_LITTLE_ADD`
`XTCODE_BOUNDED_ADD`
`XTCODE_WRAP_ADD`



### Note

Currently only `XTCODE_COMPARE_SWAP` is supported.

| | |
|---|---|
| `numArgBytes` | field to indicate the size of `argValue` in the lock request<br><br>The possible values depends on the type of lock function being requested.<br><br>**Possible Values for `numArgBytes`**<br>`0` single argument 32/64 bit `LITTLE_ADD/FETCH_ADD`<br>`4` two argument 32-bit function<br>`8` two argument 64 bit function |
| `argValue` | an array of two quadlets specifying the argument value to be used in the lock request<br><br>If `numArgBytes` is `4`, then only `argValue[0]` is used. |
| `numDataBytes` | the field indicating the size of `dataValue` in the lock request<br><br>**Possible Values for `numDataBytes`**<br>`4` single argument 32 bit `LITTLE_ADD/FETCH_ADD` two argument 32-bit function<br>`8` single argument 64 bit `LITTLE_ADD/FETCH_ADD` two argument 64 bit function |
| `dataValue` | an array of two quadlets specifying the `dataValue` to be used in the lock request<br><br>If `numDataBytes` is `4`, only `dataValue[0]` is used. |
| `buffer` | a buffer of `numDataBytes` bytes that receives the data returned in the lock transaction response |
| `ackCode` | the field in which the acknowledge code received for the read request is stored |

| | |
|---|---|
| | This field is valid only if the response was not received and `EOS_MW1394_TIMEDOUT` error is returned. |
| `respCode` | the field in which the response code contained in the response packet is saved |

### Error Values

| | |
|---|---|
| `EOS_MW1394_TIMEDOUT` | timed out waiting for response (008:260) |
| `EOS_MW1394_FAILED` | an internal failure was detected (008:261) |
| `EOS_MW1394_BUSRESET` | a 1394 bus reset was detected (008:257) |

# mw1394MapAddressRange()

Map Address Range

## Syntax

```
#include <lib1394.h>
error_code mw1394MapAddressRange(path_id path,
                        mw1394_mapaddr_pb *addrpb)
```

## Libraries

`lib1394.l`

## Description

This function maps an IEEE 1212 address range used in asynchronous requests. To use this function the caller supplies a buffer. If no buffer is supplied, the data is sent directly to the application requesting the address mapping using a callback function.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*addrpb` | pointer to the 1394 map address parameter block |

## Data Structures

The following describes the 1394 map address parameter block:

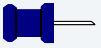
```
typedef struct mw1394_mapaddr_pb {
  void *buffer;
  u_int32 length;
  u_int32 accessType;
  u_int32 notificationOptions;
  error_code (*callback)(struct mw1394_notify_callback_pb*);
  void    *context;
  mw1394_offset desrdOffset;
  mw1394_offset assgndOffset;
} mw1394_mapaddr_pb;
```

## Structure Members

The following describes members of the 1394 map address parameter block data structure:

`buffer` — When specified, `buffer` points to the application buffer in which the asynchronous operations should be performed.

When `NULL` is specified, the `callback()` function must be provided so the IEEE 1394 stack can send the data up directly.



### Note

Under the above circumstances, the application must be in system state; it is also responsible to process the request and free the mbufs.

`Length` — specifies length of the address to map

`AccessType` — When specified, `AccessType` dictates the type of access allowed to the specified memory region.

The values can be ORed.

**Possible Values for `AccessType`**

`ACCESS_TYPE_READ`:

the memory region specified can be read by the device

`ACCESS_TYPE_WRITE`:

the memory region specified can be written to by the device

`ACCESS_TYPE_LOCK`:

a lock operation can be performed on the memory region

`notificationOptions`

When specified, `notificationOptions` dictates which type of post notification is required.

The values can be ORed. This is irrelevant if the callback field is `NULL`.

**Possible Values for `notificationOptions`**

`NOTIFY_AFTER_READ`:

notify application after asynchronous read

`NOTIFY_AFTER_WRITE`:

notify application after asynchronous write

`NOTIFTY_AFTER_LOCK`:

notify application after asynchronous lock

`callback`

points to the callback function within the application that called when the memory region was accessed

If buffer is `NULL`, this callback routine is responsible for processing the request.

See documentation for the API function, **Callback**, for information on function parameters.

# Callback
## Mapping callback function

### Syntax
```
#include <lib1394.h>
error_code (*callback)
(struct mw1394_notify_callback_pb *cbparam)
```

### Libraries
`lib1394.l`

### Description
This routine is pointed to by the `callback` parameter for `mw1394MapAddressRange()`. It points to the callback function within the application, which is called when the memory region is accessed. If the buffer is `NULL`, this routine is responsible for processing the request.

### Attributes
| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

### Parameters
| | |
|---|---|
| `cbparam` | pointer to the callback routine parameter block |

### Return Codes
| | |
|---|---|
| `RCODE_COMPLETE` | the receive operation completed successfully |
| `RCODE_DATA_ERROR` | the received data was invalid |
| `RCODE_ADDRESS_ERROR` | the received address or length was invalid |
| `RCODE_TYPE_ERROR` | the received data type was invalid |

### Data Structures

The following describes the callback parameter block:

```
typedef struct mw1394_notify_callback_pb {
  void                        *buffer;
  u_int32 offset;
  void                        **data;
  u_int32 **length;
  u_int16                     srcnode;
  u_int32                     notificationOptions;
  void  *context;
}mw1394_notify_callback_pb,*Mw1394_notify_callbac _pb;
```

### Callback Parameter Block Members

The following describes members of the callback parameter block:

| | |
|---|---|
| `buffer` | pointer to the buffer originally specified in the `map1394AddressRange` call |
| `offset` | specifies the byte offset within the buffer where the 1394 operation was requested<br>This offset is from the base of the 1394 virtual address mapped. |
| `data` | points to the address of a pointer to the buffer in which the request/response is stored<br>For an incoming asynchronous write request, the write request data is pointed at by `data`. When the incoming request is `read` or `lock`, the callback function is expected to update `date` to point at the resonse data. This field is used only for pre-notification callback. |
| `length` | points to the length in bytes of the requested 1394 operation<br>For an incoming asynchronous write request, the write request length is pointed at by length. |

| | |
|---|---|
| | If the asynchronous request is a `read` or `lock`, the callback function updates the length field to indicate the length of data returned. This field is used only in the case of pre-notification callback. |
| `srcnode` | specifies the node identifier (bus and node ID) of the node requesting the operation |
| `notificationOption` | bit that triggers the notify callback |
| `context` | points to the context data specified by the application during the `map1394AddressRange` call |

The callback routine returns a `RESPONSE CODE` used for the response code in the 1394 response packet. When using the post notifications, the callback return code (`RESPONSE CODE`) is ignored.

**Note**

Currently, the data and length fields of the callback parameter block are not updated before a callback.

| | |
|---|---|
| `context` | points to the context data for the callback routine<br><br>Callback routine is called with `context` as one fields of the callback parameter block. |
| `desrdOffset` | Unless `0X000000000000`, `desrdOffset` specifies the desired IEEE 1212 address to be used for mapping. |

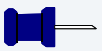
**Note**

Currently, the 1394 stack expects the `desrdOffset` field to point to a valid address; this is the address to which it is mapped.

| | |
|---|---|
| `assgndOffset` | the IEEE 1212 address mapped and assigned for the application |

### Error Values

| | |
|---|---|
| `EOS_MW1394_ADDRNOTAVAIL` | the desired address is not available for mapping (008:265) |
| `EOS_MW1394_BADSIZE` | the length of address range specified is invalid (008:262) |
| `EOS_MW1394_ADDRINUSE` | the address requested is already in use by some other application (008:261) |



**Note**

Currently, the application must specify the desired address.

# mw1394UnMapAddressRange()

Unmap Address Range

## Syntax

```
#include <lib1394.h>
error_code mw1394UnMapAddressRange(path_id path,
                                   mw1394_unmapaddr_pb *addrpb)
```

## Libraries

`lib1394.l`

## Description

This function unmaps an address range previously mapped with `mw1394MapAddressRange`.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*addrpb` | pointer to the 1394 unmap address parameter block |

## Data Structures

The following describes the 1394 unmap address parameter block:

```
typedef struct mw1394_unmapaddr_pb {
  mw1394_offset assgndOffset;
} mw1394_unmapaddr_pb;
```

## Structure Members

The following describes the members of the 1394 unmap address parameter block data structure:

| | |
|---|---|
| `assgndOffset` | the IEEE 1212 address that was mapped and assigned when the application called `mw1394MapAddressRange` |

### Error Values

| | |
|---|---|
| `EOS_MW1394_ADDRNOTFND` | the address specified was not mapped (008:264) |

# mw1394ResetBus()

Instructs Stack to Issue Bus Reset

## Syntax

```
#include <lib1394.h>
error_code mw1394ResetBus(path_id path)
```

## Libraries

`lib1394.l`

## Description

This function instructs the IEEE 1394 link layer driver to issue a bus reset.

## Attributes

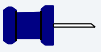
| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |

## Error Values

The link layer driver implements this function and is responsible for any error codes. Current drivers available from Microware perform a bus reset and return SUCCESS.



## Note

Issuing this command causes all existing applications to receive a BUS RESET signal from the protocol stack to indicate a bus reset

# mw1394GetTopologyMap()

Return Topology Map System

## Syntax

```
#include <lib1394.h>
error_code mw1394GetTopologyMap(path_id path,
                                mw1394topologyMap *tmap)
```

## Libraries

`lib1394.l`

## Description

This function acquires the current IEEE 1394 topology map. A call is made to retrieve this map from the bus manager, if it exists. If not, the locally maintained map is returned.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*tmap` | pointer to the buffer where the topology map will be stored |

## Error Values

| | |
|---|---|
| `EOS_MW1394_TIMEDOUT` | timed out waiting for response from the bus manager (008:260) |
| `EOS_MW1394_INVTOPOMAP` | the topology map on the bus manager is invalid (008:272) |
| `EOS_MW1394_NOIRM` | no isochronous resource manager was found (008:258) |
| `EOS_MW1394_NOBUSMGR` | a bus manager was not found (008: 287) |

**Note**

The use of this function is discouraged, since it may be replaced with more effective means of getting the bus and the node information.

# mw1394GetSpeedMap()

Return Speed Map

## Syntax

```
#include <lib1394.h>
error_code mw1394GetSpeedMap(path_id path,
                             mw1394speedMap *smap)
```

## Libraries

`lib1394.l`

## Description

This function returns the speed map of the system maintained by the bus manager. If no IRM or bus manager is found an appropriate error is returned.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*smap` | pointer to the buffer where the speed map is stored |

## Error Values

| | |
|---|---|
| `EOS_MW1394_NOIRM` | an IRM was not found (008:258) |
| `EOS_MW1394_NOBUSMGR` | a bus manager was not found (008: 287) |
| `EOS_MW1394_TIMEDOUT` | timed out waiting for response from the bus manager (008:260) |
| `EOS_MW1394_INVSPEEDMAP` | the speed map on the bus manager is invalid (008:274) |

**Note**

The use of this function is discouraged since it can be replaced by more effective means of getting the bus and node information. Moreover, the size of the speed map may be greater than the asynchronous MTU of the device, in which case the data would not be available.

# mw1394GetMaxSpeedtoNode()

## Return Speed Code

### Syntax

```
#include <lib1394.h>
error_code mw1394GetMaxSpeedtoNode(path_id path,
                                   u_int16 nodeid,
                                   u_int8 *speedcode)
```

### Libraries

`lib1394.l`

### Description

This function returns the speed code corresponding to the speed between the node and destination node maintained by the bus manager. If no IRM or bus manager is found, an appropriate error is returned.

### Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

### Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `nodeid` | identifier of the destination node |
| `*speedcode` | pointer to the location where the speed code would be returned<br>The speed code translates to speed as follows:<br>`BUS_SPEED_100MBPS` (0):<br>98.304 Mbits/second<br>`BUS_SPEED_200MBPS` (1):<br>196.608 Mbits/second |

BUS_SPEED_400MBPS (2):

393.216 Mbits/second

## Error Values

| | |
|---|---|
| EOS_MW1394_NOIRM | an IRM was not found (008:258) |
| EOS_MW1394_NOBUSMGR | a bus manager was not found (008: 287) |
| EOS_MW1394_TIMEDOUT | timed out waiting for a response from the bus manager (008:260) |
| EOS_MW1394_INVSPEEDMAP | the speed map on the bus manager is invalid (008:274) |

# mw1394BeBusManager()

Become the Bus Manager

## Syntax

```
#include <lib1394.h>
error_code mw1394BeBusManager(path_id path)
```

## Libraries

`lib1394.l`

## Description

This function instructs the 1394 stack to request the IRM to make the current node the bus manager.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

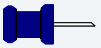
## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |

## Error Values

| | |
|---|---|
| `EOS_MW1394_NOIRM` | an IRM was not found (008:258) |
| `EOS_MW1394_BUSMGR_EXISTS` | a bus manager is present (008:275) |
| `EOS_MW1394_BADSIZE` | the length of response is invalid (008:262) |
| `EOS_MW1394_BEBUSMGRFAIL` | the request to be a bus manager failed because the IRM did not honor the request (008:276) |

| | |
|---|---|
| `EOS_MW1394_BUSMGR_ALRDY` | the requesting node is already the bus manager (008:277) |



**Note**

This command does not ensure that the node becomes the bus manager; the IRM may deny the request.

# mw1394IsochAllocateChannel()

Allocates Isochronous Channel

## Syntax

```
#include <lib1394.h>
error_code mw1394IsochAllocateChannel(path_id path,
                                      int32 *channel,
                                      quadlet *channelAvailable)
```

## Libraries

`lib1394.l`

## Description

This function allocates an isochronous channel to be used in subsequent operations.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*channel` | pointer to the locations where the caller specifies the requested channel number<br><br>`*channel` updates the location to contain the actual channel number assigned. If `CHANNEL_ANY` (`-1`) is specified, then an arbitrary channel (`-63`) is returned. |
| `*channelAvailable` | pointer to the 8-byte field containing a bit mask of the available isochronous channels after the allocation fails or succeeds |

| | |
|---|---|
| | Applications should not count on these channels being available, as another application could have allocated channels after this result is returned. |

## Error Values

| | |
|---|---|
| `EOS_MW1394_NOIRM` | an IRM was not found (008:258) |
| `EOS_MW1394_ALLOCCHNLFAIL` | the Allocation of the channel failed (008:278) |
| `EOS_MW1394_INVCHANNEL` | the channel value specified is out of range (008:266) |
| `EOS_MW1394_CHNLINUSE` | the specific channel requested is currently in use (008:267) |
| `EOS_MW1394_TIMEDOUT` | timed out waiting for response from IRM (008:260) |
| `EOS_MW1394_BADSIZE` | the length of response is invalid (008:262) |

# mw1394IsochFreeChannel()

Free Isochronous Channel

## Syntax

```
#include <lib1394.h>
error_code mw1394IsochFreeChannel(path_id path, int32 channel)
```

## Libraries

`lib1394.l`

## Description

This function frees a previously allocated isochronous channel.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `channel` | value of the channel that the caller wants to be freed<br><br>This channel should have been previously allocated by the caller using the `mw1394IsochAllocChannel` function call. |

### Error Values

| | |
|---|---|
| EOS_MW1394_NOIRM | an IRM was not found (008:258) |
| EOS_MW1394_FREECHNLFAIL | the allocation of the channel failed (008:279) |
| EOS_MW1394_INVCHANNEL | the channel value specified is out of range (008:266) |
| EOS_MW1394_CHNLFREEALRDY | the specified channel is already free (008:280) |
| EOS_MW1394_TIMEDOUT | timed out waiting for response from IRM (008:260) |

# mw1394IsochAllocateBandwidth()

Allocate Isochronous Bandwidth

## Syntax

```
#include <lib1394.h>
error_code mw1394IsochAllocateBandwidth(path_id path,
                    struct mw1394_alloc_bwdth *abwdth_pb)
```

## Libraries

`lib1394.l`

## Description

This function allocates an isochronous bandwidth to be used in subsequent operations.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*abwdth_pb` | pointer to the allocate bandwidth parameter block |

## Data Structures

The following describes the allocate bandwidth parameter block:

```
typedef struct mw1394_alloc_bwdth _pb {
  u_int32 maxBytesPerFrame;
  u_int32 speed;
  u_int32 *bandwidthHandleID;
  u_int32 *bytesPerFrameAvailable;
  u_int32 *speedSelected;
} mw1394_alloc_bwdth_pb,*Mw1394_ alloc_bwdth_pb;
```

### Structure Members

The following describes the members of the above data structure:

`maxBytesPerFrame` specifies number of bytes per isochronous frame requested

`speed` specifies speed for allocating bandwidth

**Possible Values for `speed`**

`BUS_SPEED_100MBPS`: 98.304 Mbits/second

`BUS_SPEED_200MBPS`: 196.608 Mbits/second

`BUS_SPEED_400MBPS`: 393.216 Mbits/second

`bandwidthHandleID` pointer to field containing the returned bandwidth handle ID used in releasing bandwidth resources at a later time

`bytesPerFrameAvailable`

points to the field that contains the available bytes-per-frame if allocation succeeds or fails

This bandwidth is not always available.

`speedselected` points to the field that contains the speed code selected in allocating bandwidth

**Possible Values for `speedselected`**

`BUS_SPEED_100MBPS`: 98.304 Mbits/second

`BUS_SPEED_200MBPS`: 196.608 Mbits/second

`BUS_SPEED_400MBPS`: 393.216 Mbits/second

### Error Values

| | |
|---|---|
| `EOS_MW1394_NOIRM` | an IRM was not found (008:258) |
| `EOS_MW1394_ALLOCBWDTHFAIL` | allocation of bandwidth failed (008:281) |
| `EOS_MW1394_BANDWIDTH_NOTAVAIL` | amount of bandwidth requested is not available<br>The caller may reduce the desired `maxbytesPerFrame` and try again or retry after some time. (008:283) |
| `EOS_MW1394_BADSIZE` | length of response is invalid (008:262) |
| `EOS_MW1394_TIMEDOUT` | timed out waiting for response from IRM (008:260) |

# mw1394IsochFreeBandwidth()

Free Isochronous Bandwidth

## Syntax

```
#include <lib1394.h>
error_code mw1394IsochFreeBandwidth(path_id path,
                                    u_       int32 handleID)
```

## Libraries

`lib1394.l`

## Description

This function frees a previously allocated isochronous bandwidth.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `handleID` | handle identifier of the bandwidth that the caller wants to be freed<br><br>This bandwidth should have been previously allocated by the caller using the `mw1394IsochAllocBandwidth` function call. |

### Error Values

| | |
|---|---|
| `EOS_MW1394_NOIRM` | an IRM was not found (008:258) |
| `EOS_MW1394_INV_BANDWIDTH_HNDL` | no bandwidth handle is found with the specified handleID (008:284) |
| `EOS_MW1394_TIMEDOUT` | timed out waiting for response from IRM (008:260) |

# mw1394IsochAttachBuffers()

Attach Isochronous Buffers

## Syntax

```
#include <lib1394.h>
mw1394IsochAttachBuffers(path_id path,
                         struct mw1394_isochdesc_pb *isodesc)
```

## Libraries

`lib1394.l`

## Description

This function attaches isochronous buffers to the specified path.

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*isodesc` | pointer to the isochronous channel descriptor parameter block |

## Data Structures

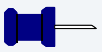
Following is the isochronous channel descriptor parameter block:

```
typedef struct mw1394_isochdesc_ pb {
  u_int32 channel;
  quadlet *buffer;
  u_int32 bufSize;
  u_int32 synchronize;
  u_int32 flags;
  u_int32 cycle;
  error_code (*callback)(void *);
  error_code (*waterMarkCallback) (void *);
  u_int32 waterMark;
  void    *context;
  u_int32 status;
  u_int32 pktSize;
  u_int32 rsvd[4];
} mw1394_isochdesc_pb, *Mw1394_ isochdesc_pb;
```

### Structure Members

The following describes the members of the isochronous channel descriptor parameter block data structure:

| | |
|---|---|
| `channel` | channel number to which this buffer is to be attached |
| `buffer` | pointer to an isochronous buffer used with this channel in which data is contained |
| `bufSize` | the length of the buffer |
| `synchronize` | used to synchronize packet acceptance with the `sy` field of the 1394 isochronous packet header |
| `flags` | bit flags used for synchronizing packet acceptance and packet header removal before moving the data to the buffer<br>**Valid Bit Fields for `flags`**<br>`FLAG_SYNCHRONIZE: 0x01`<br>`FLAG_STRIP_HEADER: 0x02`<br>`FLAG_STRIP_CIP_HDR: 0x08` |
| `cycle` | not currently used |
| `callback` | specifies the callback routine that is called when the requested data is available<br>Applications can then be notified of the availability of the data. |



**Note**

This is only available for reception of isochronous data.

| | |
|---|---|
| `waterMarkCallback` | specifies the callback routine that is called when the `waterMark` is reached |

## Note

The watermark callback is only available for reception of isochronous data.

| | |
|---|---|
| `waterMark` | specifies the minimum amount of data to be present in the buffer at which the `waterMarkCallback` routine is invoked |
| `context` | the user supplied context parameter to be provided at callback time |
| `status` | not currently used |
| `pktSize` | the size of the packet to receive or transfer<br>The value is specified in bytes |

### Error Values

| | |
|---|---|
| `EOS_MW1394_BADSIZE` | the packet size specified is greater than the maximum allowable isochronous payload at the PHY speed (008:262) |

# mw1394IsochDetachBuffers()

Detach Isochronous Buffers from Path

## Syntax

```
#include <lib1394.h>
mw1394IsochDetachBuffers(path_id path, u_int32 channel)
```

## Libraries

`lib1394.l`

## Description

This function attaches isochronous buffers to the specified path.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `channel` | channel number associated with the buffers attached |

## Error Values

| | |
|---|---|
| `EOS_MW1394_NOBUFFERS_ATTCHD` | no buffers attached for the specified channel (008:285) |

# mw1394IsochListen()

Listen on Isochronous Channel

## Syntax

```
#include <lib1394.h>
mw1394IsochListen(path_id path,struct mw1394_isoch_pb *isochpb)
```

## Libraries

`lib1394.l`

## Description

Begin listening on a specified isochronous channel.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*Isochpb` | pointer to isochronous parameter block |

## Data Structures

The following describes the isochronous parameter block:

```
typedef struct mw1394_isoch_pb {
  u_int32 channel;
  u_int32 isoFlags;
  u_int32 startCycle;
  u_int32 startTime;
  u_int32 synchronize;
  u_int32 tag;
} mw1394_isoch_pb, *Mw1394_isoch_pb;
```

### Structure Members

The following describes the members of the isochronous parameter block data structure:

| | |
|---|---|
| `channel` | specifies the channel on which to listen |
| `isoFlags` | not currently used |
| `startCycle` | not currently used |
| `startTime` | not currently used |
| `synchronize` | not currently used |
| `tag` | not currently used |

### Error Values

| | |
|---|---|
| `EOS_MW1394_NOBUFFERS_ATTCHD` | no buffers attached for the specified channel (008:285) |
| `EOS_MW1394_INVCHANNEL` | the channel value specified is out of range (008:266) |
| `EOS_MW1394_CHNLINUSE` | the specific channel requested is currently in use (008:267) |
| `EOS_MW1394_NOCFGREG` | no free isochronous configuration registers are available (008:268) |

# mw1394IsochXmit()

Transmit on Isochronous Channel

## Syntax

```
#include <lib1394.h>
mw1394IsochXmit(path_id path, struct mw1394_isoch_pb *isochpb)
```

## Libraries

`lib1394.l`

## Description

Begin transmitting on the specified channel.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*Isochpb` | pointer to isochronous parameter block |

## Data Structures

The following describes the isochronous parameter block:

```
typedef struct mw1394_isoch_pb {
  u_int32 channel;
  u_int32 isoFlags;
  u_int32 startCycle;
  u_int32 startTime;
  u_int32 synchronize;
  u_int32 tag;
} mw1394_isoch_pb, *Mw1394_isoch_pb;
```

### Structure Members

The following describes members of the isochronous parameter block data structures:

| | |
|---|---|
| `channel` | specifies channel on which to transmit |
| `isoFlags` | not currently used |
| `startCycle` | not currently used |
| `startTime` | not currently used |
| `synchronize` | not currently used |
| `tag` | set tag in isochronous packet header |

### Error Values

`EOS_MW1394_NOBUFFERS_ATTCHD`

no buffers attached for the specified channel (008:285)

`EOS_MW1394_CHNL_STOPPED`

operations on this channel have been stopped by a call to `ms1394IsochStop` (008:270)

# mw1394IsochStop()

Stop Operations on Isochronous Channel

## Syntax

```
#include <lib1394.h>
mw1394IsochStop(path_id path, struct mw1394_isoch_pb *isochpb)
```

## Libraries

`lib1394.l`

## Description

This function stops operations on the specified isochronous channel.

## Attributes

| | |
|---|---|
| Operating System: | OS-9 |
| State: | User |
| Threads: | Safe |

## Parameters

| | |
|---|---|
| `path` | path id of the path opened at init time |
| `*isochpb` | pointer to isochronous parameter block |

## Data Structures

The following describes the isochronous parameter block:

```
typedef struct mw1394_isoch_pb {
  u_int32 channel;
  u_int32 isoFlags;
  u_int32 startCycle;
  u_int32 startTime;
  u_int32 synchronize;
  u_int32 tag;
} mw1394_isoch_pb, *Mw1394_isoch_pb;
```

### Structure Members

The following describes members of the isochronous parameter block data structure:

| | |
|---|---|
| `channel` | specifies the channel to stop operations |
| `isoFlags` | not currently used |
| `startCycle` | not currently used |
| `startTime` | not currently used |
| `synchronize` | not currently used |
| `tag` | not currently used |

### Error Values

| | |
|---|---|
| `EOS_MW1394_NOBUFFERS_ATTCHD` | no buffers attached for the specified channel (008:285) |
| `EOS_MW1394_INVCHANNEL` | the channel value specified is out of range (008:266) |
| `EOS_MW1394_CHNLNOTFND` | the specified channel was not found in the isochronous configuration register (008:269) |

**Note**

Currently, `mwIsochStop` can be performed only on channels to which the application is listening.

# Transfer Sequence

The following lists represent the sequences of calls recommended for the use of the IEEE 1394 stack for either asynchronous or isochronous data transfer.

## Asynchronous Transfer Sequence

- `mw1394Initialize`
- `mw1394MapAddressRange`
- `mw1394AsynchWrite, mw1394AsynchRead, mw1394AsynchLock, mw1394AsynchBlockWrite`
- `mw1394UnmapAddressRange`
- `mw1394Terminate`

## Isochronous Transfer Sequence

- `mw1394Initialize`
- `mw1394IsochAllocateChannel`
- `mw1394IsochAllocateBandwidth`
- `mw1394IsochAttachBuffers`
- `mw1394IsochListen/mw1394IsochXmit`
- `mw1394IsochStop:` Optional
- `mw1394IsochDetachBuffers`
- `mw1394IsochFreeBandwidth`
- `mw1394IsochFreeChannel`
- `mw1394Terminate`

# Product Discrepancy Report

To: Microware Customer Support

FAX: 515-224-1352

From:_____

Company:_____

Phone:_____

Fax:_____Email:_____

Product Name:

Description of Problem:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Host Platform_____

Target Platform_____