ChorusOS man pages section 2DL: Data Link Services

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 806-3325
December 10, 1999

Contents

PREFACE

Overview

A man page is provided for both the naive user, and sophisticated user who is familiar with the ChorusOS™ operating system and is in need of on-line information. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

The following is a list of sections in the ChorusOS man pages and the information it references:

- *Section 1CC: User Utilities; Host and Target Utilities*
- *Section 1M: System Management Utilities*
- *Section 2DL: System Calls; Data Link Services*
- *Section 2K: System Calls; Kernel Services*
- *Section 2MON: System Calls; Monitoring Services*
- *Section 2POSIX: System Calls; POSIX System Calls*
- *Section 2RESTART: System Calls; Hot Restart and Persistent Memory*
- *Section 2SEG: System Calls; Virtual Memory Segment Services*
- *Section 3FTPD: Libraries; FTP Daemon*
- *Section 3M: Libraries; Mathematical Libraries*
- *Section 3POSIX: Libraries; POSIX Library Functions*
- *Section 3RPC: Libraries; RPC Services*
- *Section 3STDC: Libraries; Standard C Library Functions*
- *Section 3TELD: Libraries; Telnet Services*
- *Section 4CC: Files*

- *Section 5FEA: ChorusOS Features and APIs*
- *Section 7P: Protocols*
- *Section 7S: Services*
- *Section 9DDI: Device Driver Interfaces*
- *Section 9DKI: Driver to Kernel Interface*
- *Section 9DRV: Driver Implementations*

ChorusOS man pages are grouped in Reference Manuals, with one reference manual per section.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and `man`(1) for more information about man pages in general.

NAME — This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS — This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

. . . Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, ‘ "`filename` . . ." .

| Separator. Only one of the arguments separated by this character can be specified at time.

{ } Braces. The options and / or arguments enclosed within braces are

	interdependent, such that everything enclosed must be treated as a unit.
FEATURES	This section provides the list of features which offer an interface. An API may be associated with one or more system features. The interface will be available if one of the associated features has been configured.
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES.. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.
OPTIONS	This lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output - standard output, standard error, or output files - generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

USAGE	This section is provided as a guidance on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality: Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%` or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.
ENVIRONMENT VARIABLES	This section lists any environment variables that the command or function affects, followed by a brief description of the effect.
EXIT STATUS	This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion and values other than zero for various error conditions.
FILES	This section lists all filenames referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.
SEE ALSO	This section lists references to other man pages, in-house documentation and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.

BUGS

This section describes known bugs and wherever possible, suggests workarounds.

CHAPTER

Data Link Services

10

NAME

svDataLink, svDataLinkAttach, svOutFrameFree, svInputFrameDeliver – Attach a Chorus IPC Data Link Driver; Free an outgoing frame; Deliver an incoming frame

SYNOPSIS

```
#include <ipc/extDtLink.h>
int svDataLinkAttach(ExtDtLink * dtLink);

void svOutFrameFree(CnOutFrame * outFrame);

void svInputFrameDeliver(ExtDtLink * dtLink, CnInFrame * inputFr);
```

FEATURES

IPC_REMOTE

DESCRIPTION

ChorusOS includes an implementation of remote Chorus IPC over Ethernet (through the DATALINK_INET feature, when combined with ETHERNET). The system calls described below allow users to implement the transmission of Chorus IPC messages over other network types. To implement Chorus IPC over a new network medium, perform the following steps:

- Unset any DATALINK_ features from the system configuration.
- Load a supervisor actor which declares itself to the ChorusOS kernel using the *svDataLinkAttach* system call. This actor is called a *data link driver* .

The primary function of a data link driver is to transmit Chorus IPC message *frames* between Chorus sites, in both unicast and broadcast modes. The maximum frame size is defined by the data link driver, rather than being imposed by the ChorusOS kernel. The only transmission guarantee expected from the data link driver is frame integrity (the data link driver must insure that the frame contents are preserved during transmission). In addition, to ensure proper Chorus IPC performance, the data link driver should transmit every frame and maintain a FIFO ordering. Disordered or lost frames are tolerated by IPC protocols, but should be avoided.

svDataLinkAttach function registers a new data link driver. The *dtLink* parameter is a pointer to an *ExtDtLink* structure whose members are the following:

```
typedef struct ExtDtLink_t {
     void*         cookie;         /* Reserved - only used by the kernel */
     char*         dtLinkName;     /* Data link driver name */
     unsigned int  frameHdrSize;   /* e.g. 14 for Ethernet */
     unsigned int  maxFrameSize;   /* Must include frameHdrSize */
     FrameSend     frameSend;      /* To a particular remote site */
     FrameSend     frameBcast;     /* To all reachable sites */
} ExtDtLink;
```

The data link driver sends the following information to the ChorusOS kernel:

- The name of the data link driver as a character string, pointed to by *dtLinkName* .
- The size of its frame header,expressed in bytes in *frameHdrSize;* this information will allow the kernel to allocate room for the data link header within each frame.
- The maximum frame size (including the frame header), expressed in bytes in *maxframeSize*
- The function which the kernel will invoke when sending a unicast frame, in *frameSend* .
- The function which the kernel will invoke when sending a broadcast frame, in *frameBcast* .

Both *frameSend* and *frameBcast* are pointers to functions whose arguments are the following:

```
void frameSend (
     CnOutFrame*   frame,
     ExtDtLink*    dtLink);
```

The *dtLink* parameter is a pointer to the *ExtDtLink* structure declared by the data link driver when it attaches itself. The *frame* parameter is a pointer to a *CnOutFrame* structure, which describes the frame to be sent, as follows:

```
typedef struct CnOutFrame_t {
     struct CnOutFrame_t*   next;
     unsigned int           totalLength;
     MemBuffer*             bufList;
     unsigned int           destSite;
} CnOutFrame;
```

The *destSite* parameter identifies the Chorus site number to the *frameSend* function. When sent to the *frameBcast* function, *destSite* is set to 0xFFFFFFFF. This allows data link drivers to implement a single function, and to check for broadcast mode from the destination site number.

The *bufList* parameter is the first of a single-linked list of *MemBuffer* structures which describe the memory buffers holding the frame data, as follows:

```
typedef struct MemBuffer_t {
     struct MemBuffer_t*   next;
     char*                 address;
     unsigned int          size;
} MemBuffer;
```

The *next* pointer indicates the next *MemBuffer* on the list, and is NULL in the last buffer. The *address* pointer indicates the first byte of the memory buffer, and `size` is the size of the memory buffer, expressed in bytes.

The total frame size is given by *totalLength*, and is assumed to be lower than or equal to the *maxFrameSize* field of *dtLink*.

When a frame is passed to the data link driver, the space for storing the data link header has been reserved at the beginning of the first memory buffer. The `size` field of the first memory buffer, as well as the *totalLength* field of the frame descriptor both include the size of the data link header.

When the data link driver is invoked to send a frame, it should perform the following functions:

- Resolve the address(es) of the destination node(s) from the Chorus site number (*destSite*).
- Update its header within the frame.
- Send or broadcast the frame.

The *frameSend* function may be invoked from an interrupt (time-out handler) by the kernel.

When a frame has been sent, the data link driver must invoke *svOutFrameFree* in order to notify the kernel that the frame data can be freed.

When receiving a frame from the network, the data link driver must invoke the *svInputFrameDeliver* system call. The *dtLink* parameter is a pointer to the data link descriptor, *inputFr* is a pointer to the *CnInFrame* structure, which describes the frame received, as follows:

```
typedef struct CnInFrame_t {
     struct CnInFrame_t*   next;
     unsigned int          totalLength;
     MemBuffer*            bufList;
} CnInFrame;
```

The *next* , *totalLength* and *bufList* have the same meanings as in the *CnOutFrame* structure.

The *svInputFrameDeliver* function is intended to be invoked from an interrupt. Upon return from *svInputFrameDeliver*, the data described by *inputFr* has been copied by the kernel into receiver memory, and the data link driver can reuse it (for example, put it back into a network controller receive ring).

RETURN VALUE

The *svDataLinkAttach* function returns a value of 0 when successfully completed. Otherwise, a negative error code is returned.

ERRORS

[K_EINVAL] A data link driver has already declared itself.

[K_ENOMEM] The system is out of resources.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

svDataLink, svDataLinkAttach, svOutFrameFree, svInputFrameDeliver – Attach a Chorus IPC Data Link Driver; Free an outgoing frame; Deliver an incoming frame

SYNOPSIS

#include <ipc/extDtLink.h>
int **svDataLinkAttach**(ExtDtLink * *dtLink*);

void **svOutFrameFree**(CnOutFrame * *outFrame*);

void **svInputFrameDeliver**(ExtDtLink * *dtLink*, CnInFrame * *inputFr*);

FEATURES

IPC_REMOTE

DESCRIPTION

ChorusOS includes an implementation of remote Chorus IPC over Ethernet (through the DATALINK_INET feature, when combined with ETHERNET). The system calls described below allow users to implement the transmission of Chorus IPC messages over other network types. To implement Chorus IPC over a new network medium, perform the following steps:

- Unset any DATALINK_ features from the system configuration.
- Load a supervisor actor which declares itself to the ChorusOS kernel using the *svDataLinkAttach* system call. This actor is called a *data link driver* .

The primary function of a data link driver is to transmit Chorus IPC message *frames* between Chorus sites, in both unicast and broadcast modes. The maximum frame size is defined by the data link driver, rather than being imposed by the ChorusOS kernel. The only transmission guarantee expected from the data link driver is frame integrity (the data link driver must insure that the frame contents are preserved during transmission). In addition, to ensure proper Chorus IPC performance, the data link driver should transmit every frame and maintain a FIFO ordering. Disordered or lost frames are tolerated by IPC protocols, but should be avoided.

svDataLinkAttach function registers a new data link driver. The *dtLink* parameter is a pointer to an *ExtDtLink* structure whose members are the following:

```
typedef struct ExtDtLink_t {
     void*         cookie;         /* Reserved - only used by the kernel */
     char*         dtLinkName;     /* Data link driver name */
     unsigned int  frameHdrSize;   /* e.g. 14 for Ethernet */
     unsigned int  maxFrameSize;   /* Must include frameHdrSize */
     FrameSend     frameSend;      /* To a particular remote site */
     FrameSend     frameBcast;     /* To all reachable sites */
} ExtDtLink;
```

The data link driver sends the following information to the ChorusOS kernel:

- The name of the data link driver as a character string, pointed to by *dtLinkName* .
- The size of its frame header,expressed in bytes in *frameHdrSize;* this information will allow the kernel to allocate room for the data link header within each frame.
- The maximum frame size (including the frame header), expressed in bytes in *maxframeSize*
- The function which the kernel will invoke when sending a unicast frame, in *frameSend* .
- The function which the kernel will invoke when sending a broadcast frame, in *frameBcast* .

Both *frameSend* and *frameBcast* are pointers to functions whose arguments are the following:

```
void frameSend (
     CnOutFrame*   frame,
     ExtDtLink*    dtLink);
```

The *dtLink* parameter is a pointer to the *ExtDtLink* structure declared by the data link driver when it attaches itself. The *frame* parameter is a pointer to a *CnOutFrame* structure, which describes the frame to be sent, as follows:

```
typedef struct CnOutFrame_t {
     struct CnOutFrame_t*   next;
     unsigned int           totalLength;
     MemBuffer*             bufList;
     unsigned int           destSite;
} CnOutFrame;
```

The *destSite* parameter identifies the Chorus site number to the *frameSend* function. When sent to the *frameBcast* function, *destSite* is set to 0xFFFFFFFF. This allows data link drivers to implement a single function, and to check for broadcast mode from the destination site number.

The *bufList* parameter is the first of a single-linked list of *MemBuffer* structures which describe the memory buffers holding the frame data, as follows:

```
typedef struct MemBuffer_t {
     struct MemBuffer_t*   next;
     char*                 address;
     unsigned int          size;
} MemBuffer;
```

The *next* pointer indicates the next *MemBuffer* on the list, and is NULL in the last buffer. The *address* pointer indicates the first byte of the memory buffer, and `size` is the size of the memory buffer, expressed in bytes.

The total frame size is given by *totalLength*, and is assumed to be lower than or equal to the *maxFrameSize* field of *dtLink*.

When a frame is passed to the data link driver, the space for storing the data link header has been reserved at the beginning of the first memory buffer. The `size` field of the first memory buffer, as well as the *totalLength* field of the frame descriptor both include the size of the data link header.

When the data link driver is invoked to send a frame, it should perform the following functions:

- Resolve the address(es) of the destination node(s) from the Chorus site number (*destSite*).
- Update its header within the frame.
- Send or broadcast the frame.

The *frameSend* function may be invoked from an interrupt (time-out handler) by the kernel.

When a frame has been sent, the data link driver must invoke *svOutFrameFree* in order to notify the kernel that the frame data can be freed.

When receiving a frame from the network, the data link driver must invoke the *svInputFrameDeliver* system call. The *dtLink* parameter is a pointer to the data link descriptor, *inputFr* is a pointer to the *CnInFrame* structure, which describes the frame received, as follows:

```
typedef struct CnInFrame_t {
     struct CnInFrame_t*   next;
     unsigned int          totalLength;
     MemBuffer*            bufList;
} CnInFrame;
```

The *next* , *totalLength* and *bufList* have the same meanings as in the *CnOutFrame* structure.

The *svInputFrameDeliver* function is intended to be invoked from an interrupt. Upon return from *svInputFrameDeliver*, the data described by *inputFr* has been copied by the kernel into receiver memory, and the data link driver can reuse it (for example, put it back into a network controller receive ring).

RETURN VALUE

The *svDataLinkAttach* function returns a value of 0 when successfully completed. Otherwise, a negative error code is returned.

ERRORS

[K_EINVAL]	A data link driver has already declared itself.
[K_ENOMEM]	The system is out of resources.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

svDataLink, svDataLinkAttach, svOutFrameFree, svInputFrameDeliver – Attach a Chorus IPC Data Link Driver; Free an outgoing frame; Deliver an incoming frame

SYNOPSIS

```
#include <ipc/extDtLink.h>
int svDataLinkAttach(ExtDtLink * dtLink);

void svOutFrameFree(CnOutFrame * outFrame);

void svInputFrameDeliver(ExtDtLink * dtLink, CnInFrame * inputFr);
```

FEATURES

IPC_REMOTE

DESCRIPTION

ChorusOS includes an implementation of remote Chorus IPC over Ethernet (through the DATALINK_INET feature, when combined with ETHERNET). The system calls described below allow users to implement the transmission of Chorus IPC messages over other network types. To implement Chorus IPC over a new network medium, perform the following steps:

- Unset any DATALINK_ features from the system configuration.
- Load a supervisor actor which declares itself to the ChorusOS kernel using the *svDataLinkAttach* system call. This actor is called a *data link driver* .

The primary function of a data link driver is to transmit Chorus IPC message *frames* between Chorus sites, in both unicast and broadcast modes. The maximum frame size is defined by the data link driver, rather than being imposed by the ChorusOS kernel. The only transmission guarantee expected from the data link driver is frame integrity (the data link driver must insure that the frame contents are preserved during transmission). In addition, to ensure proper Chorus IPC performance, the data link driver should transmit every frame and maintain a FIFO ordering. Disordered or lost frames are tolerated by IPC protocols, but should be avoided.

svDataLinkAttach function registers a new data link driver. The *dtLink* parameter is a pointer to an *ExtDtLink* structure whose members are the following:

```
typedef struct ExtDtLink_t {
     void*         cookie;         /* Reserved - only used by the kernel */
     char*         dtLinkName;     /* Data link driver name */
     unsigned int  frameHdrSize;   /* e.g. 14 for Ethernet */
     unsigned int  maxFrameSize;   /* Must include frameHdrSize */
     FrameSend     frameSend;      /* To a particular remote site */
     FrameSend     frameBcast;     /* To all reachable sites */
} ExtDtLink;
```

The data link driver sends the following information to the ChorusOS kernel:

- The name of the data link driver as a character string, pointed to by *dtLinkName* .
- The size of its frame header,expressed in bytes in *frameHdrSize;* this information will allow the kernel to allocate room for the data link header within each frame.
- The maximum frame size (including the frame header), expressed in bytes in *maxframeSize*
- The function which the kernel will invoke when sending a unicast frame, in *frameSend* .
- The function which the kernel will invoke when sending a broadcast frame, in *frameBcast* .

Both *frameSend* and *frameBcast* are pointers to functions whose arguments are the following:

```
void frameSend (
     CnOutFrame*   frame,
     ExtDtLink*    dtLink);
```

The *dtLink* parameter is a pointer to the *ExtDtLink* structure declared by the data link driver when it attaches itself. The *frame* parameter is a pointer to a *CnOutFrame* structure, which describes the frame to be sent, as follows:

```
typedef struct CnOutFrame_t {
     struct CnOutFrame_t*   next;
     unsigned int           totalLength;
     MemBuffer*             bufList;
     unsigned int           destSite;
} CnOutFrame;
```

The *destSite* parameter identifies the Chorus site number to the *frameSend* function. When sent to the *frameBcast* function, *destSite* is set to 0xFFFFFFFF. This allows data link drivers to implement a single function, and to check for broadcast mode from the destination site number.

The *bufList* parameter is the first of a single-linked list of *MemBuffer* structures which describe the memory buffers holding the frame data, as follows:

```
typedef struct MemBuffer_t {
     struct MemBuffer_t*   next;
     char*                 address;
     unsigned int          size;
} MemBuffer;
```

The *next* pointer indicates the next *MemBuffer* on the list, and is NULL in the last buffer. The *address* pointer indicates the first byte of the memory buffer, and `size` is the size of the memory buffer, expressed in bytes.

The total frame size is given by *totalLength*, and is assumed to be lower than or equal to the *maxFrameSize* field of *dtLink*.

When a frame is passed to the data link driver, the space for storing the data link header has been reserved at the beginning of the first memory buffer. The `size` field of the first memory buffer, as well as the *totalLength* field of the frame descriptor both include the size of the data link header.

When the data link driver is invoked to send a frame, it should perform the following functions:

- Resolve the address(es) of the destination node(s) from the Chorus site number (*destSite*).
- Update its header within the frame.
- Send or broadcast the frame.

The *frameSend* function may be invoked from an interrupt (time-out handler) by the kernel.

When a frame has been sent, the data link driver must invoke *svOutFrameFree* in order to notify the kernel that the frame data can be freed.

When receiving a frame from the network, the data link driver must invoke the *svInputFrameDeliver* system call. The *dtLink* parameter is a pointer to the data link descriptor, *inputFr* is a pointer to the *CnInFrame* structure, which describes the frame received, as follows:

```
typedef struct CnInFrame_t {
     struct CnInFrame_t*   next;
     unsigned int          totalLength;
     MemBuffer*            bufList;
} CnInFrame;
```

The *next* , *totalLength* and *bufList* have the same meanings as in the *CnOutFrame* structure.

The *svInputFrameDeliver* function is intended to be invoked from an interrupt. Upon return from *svInputFrameDeliver*, the data described by *inputFr* has been copied by the kernel into receiver memory, and the data link driver can reuse it (for example, put it back into a network controller receive ring).

RETURN VALUE

The *svDataLinkAttach* function returns a value of 0 when successfully completed. Otherwise, a negative error code is returned.

ERRORS

[K_EINVAL]	A data link driver has already declared itself.
[K_ENOMEM]	The system is out of resources.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

svDataLink, svDataLinkAttach, svOutFrameFree, svInputFrameDeliver – Attach a Chorus IPC Data Link Driver; Free an outgoing frame; Deliver an incoming frame

SYNOPSIS

#include <ipc/extDtLink.h>
int **svDataLinkAttach**(ExtDtLink * *dtLink*);

void **svOutFrameFree**(CnOutFrame * *outFrame*);

void **svInputFrameDeliver**(ExtDtLink * *dtLink*, CnInFrame * *inputFr*);

FEATURES

IPC_REMOTE

DESCRIPTION

ChorusOS includes an implementation of remote Chorus IPC over Ethernet (through the DATALINK_INET feature, when combined with ETHERNET). The system calls described below allow users to implement the transmission of Chorus IPC messages over other network types. To implement Chorus IPC over a new network medium, perform the following steps:

- Unset any DATALINK_ features from the system configuration.
- Load a supervisor actor which declares itself to the ChorusOS kernel using the *svDataLinkAttach* system call. This actor is called a *data link driver* .

The primary function of a data link driver is to transmit Chorus IPC message *frames* between Chorus sites, in both unicast and broadcast modes. The maximum frame size is defined by the data link driver, rather than being imposed by the ChorusOS kernel. The only transmission guarantee expected from the data link driver is frame integrity (the data link driver must insure that the frame contents are preserved during transmission). In addition, to ensure proper Chorus IPC performance, the data link driver should transmit every frame and maintain a FIFO ordering. Disordered or lost frames are tolerated by IPC protocols, but should be avoided.

svDataLinkAttach function registers a new data link driver. The *dtLink* parameter is a pointer to an *ExtDtLink* structure whose members are the following:

```
typedef struct ExtDtLink_t {
     void*         cookie;         /* Reserved - only used by the kernel */
     char*         dtLinkName;     /* Data link driver name */
     unsigned int  frameHdrSize;   /* e.g. 14 for Ethernet */
     unsigned int  maxFrameSize;   /* Must include frameHdrSize */
     FrameSend     frameSend;      /* To a particular remote site */
     FrameSend     frameBcast;     /* To all reachable sites */
} ExtDtLink;
```

The data link driver sends the following information to the ChorusOS kernel:

- The name of the data link driver as a character string, pointed to by *dtLinkName* .
- The size of its frame header,expressed in bytes in *frameHdrSize;* this information will allow the kernel to allocate room for the data link header within each frame.
- The maximum frame size (including the frame header), expressed in bytes in *maxframeSize*
- The function which the kernel will invoke when sending a unicast frame, in *frameSend* .
- The function which the kernel will invoke when sending a broadcast frame, in *frameBcast* .

Both *frameSend* and *frameBcast* are pointers to functions whose arguments are the following:

```
void frameSend (
     CnOutFrame*   frame,
     ExtDtLink*    dtLink);
```

The *dtLink* parameter is a pointer to the *ExtDtLink* structure declared by the data link driver when it attaches itself. The *frame* parameter is a pointer to a *CnOutFrame* structure, which describes the frame to be sent, as follows:

```
typedef struct CnOutFrame_t {
     struct CnOutFrame_t*   next;
     unsigned int           totalLength;
     MemBuffer*             bufList;
     unsigned int           destSite;
} CnOutFrame;
```

The *destSite* parameter identifies the Chorus site number to the *frameSend* function. When sent to the *frameBcast* function, *destSite* is set to 0xFFFFFFFF. This allows data link drivers to implement a single function, and to check for broadcast mode from the destination site number.

The *bufList* parameter is the first of a single-linked list of *MemBuffer* structures which describe the memory buffers holding the frame data, as follows:

```
typedef struct MemBuffer_t {
     struct MemBuffer_t*   next;
     char*                 address;
     unsigned int          size;
} MemBuffer;
```

The *next* pointer indicates the next *MemBuffer* on the list, and is NULL in the last buffer. The *address* pointer indicates the first byte of the memory buffer, and `size` is the size of the memory buffer, expressed in bytes.

The total frame size is given by *totalLength*, and is assumed to be lower than or equal to the *maxFrameSize* field of *dtLink*.

When a frame is passed to the data link driver, the space for storing the data link header has been reserved at the beginning of the first memory buffer. The `size` field of the first memory buffer, as well as the *totalLength* field of the frame descriptor both include the size of the data link header.

When the data link driver is invoked to send a frame, it should perform the following functions:

- Resolve the address(es) of the destination node(s) from the Chorus site number (*destSite*).
- Update its header within the frame.
- Send or broadcast the frame.

The *frameSend* function may be invoked from an interrupt (time-out handler) by the kernel.

When a frame has been sent, the data link driver must invoke *svOutFrameFree* in order to notify the kernel that the frame data can be freed.

When receiving a frame from the network, the data link driver must invoke the *svInputFrameDeliver* system call. The *dtLink* parameter is a pointer to the data link descriptor, *inputFr* is a pointer to the *CnInFrame* structure, which describes the frame received, as follows:

```
typedef struct CnInFrame_t {
     struct CnInFrame_t*   next;
     unsigned int          totalLength;
     MemBuffer*            bufList;
} CnInFrame;
```

The *next* , *totalLength* and *bufList* have the same meanings as in the *CnOutFrame* structure.

The *svInputFrameDeliver* function is intended to be invoked from an interrupt. Upon return from *svInputFrameDeliver*, the data described by *inputFr* has been copied by the kernel into receiver memory, and the data link driver can reuse it (for example, put it back into a network controller receive ring).

RETURN VALUE

The *svDataLinkAttach* function returns a value of 0 when successfully completed. Otherwise, a negative error code is returned.

ERRORS

[K_EINVAL] A data link driver has already declared itself.

[K_ENOMEM] The system is out of resources.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

Index