Simpact STREAMS ICP2432 Programmer's Guide for Windows NT®

DC 900-1544B

Simpact, Inc.
9210 Sky Park Court
San Diego, CA 92123
June 1998

SIMPACT

Simpact, Inc.
9210 Sky Park Court
San Diego, CA 92123
(619) 565-1865

Simpact STREAMS ICP2432 Programmer's Guide for Windows NT

Printed in the United States of America

Contents

List of Figures

List of Tables

Preface

Purpose of Document

This document describes how to configure and initialize Simpact STREAMS and how to write STREAMS applications for Windows NT. It also describes the internal operation of the Simpact STREAMS application programming interface and STREAMS manager task.

Intended Audience

This document is intended primarily for Windows NT system managers and applications programmers. Applications programmers should know how to develop applications in the Windows NT environment and should be familiar with the standard STREAMS application interface.

Note

Programmers who will be developing STREAMS modules and drivers for execution on the ICP2432 should refer to MetaSphere's *Embedded STREAMS Programmer's Guide & Reference* as well as Chapter 4 of this manual.

Organization of Document

Chapter 1 describes the Simpact STREAMS architecture.

Chapter 2 describes the interface to the Simpact STREAMS application program interface.

Chapter 3 describes the messages passed between the STREAMS API and the ICP's Streaman task.

Chapter 4 describes the Simpact STREAMS development environment.

References

- *Embedded STREAMS Programmer's Guide & Reference* MetaSphere, Inc.
- *Freeway OS/Impact Programmer's Guide* DC 900-1030
- *ICP2432 Hardware Installation Guide* DC 900-1502
- *ICP2432 User's Guide for Windows NT* DC 900-1510

Document Revision History

The revision history of the *Simpact STREAMS ICP2432 Programmer's Guide for Windows NT*, Simpact document DC 900-1544B, is recorded below:

Document Revision	Release Date	Description
DC 900-1544A	April 1998	Original release
DC 900-1544B	June 1998	File name change: strasm.asm to sample.asm

Customer Support

If you are having trouble with any Simpact product, call us at 1-800-275-3889 Monday through Friday between 8 a.m. and 5 p.m. Pacific time.

You can also fax your questions to us at (619)560-2838 or (619)560-2837 any time. Please include a cover sheet addressed to "Customer Service."

We are always interested in suggestions for improving our products. You can use the report form in the back of this manual to send us your recommendations.

Chapter

1 Simpact STREAMS Architecture

Simpact's typical development environment on the ICP2432 is provided by OS/Impact, a task-oriented real-time operating system. Communication with the host system is provided by the Simpact XIO interface, which transfers data to and from a host device driver using "node numbers" to identify individual connections. Figure 1–1 illustrates this environment.

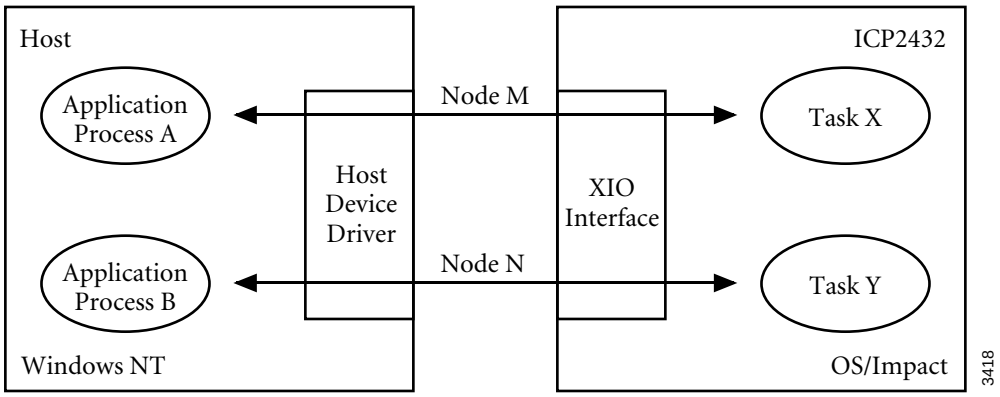

Figure 1–1: Standard Simpact Environment

Embedded STREAMS from MetaSphere is an emulation package for real-time kernels providing a STREAMS environment similar to that found in a UNIX system. In this environment, STREAMS runs as a sub-system of the real-time kernel, managed by a STREAMS scheduler task (provided as part of the Embedded STREAMS package). An application process on the embedded system (an OS/Impact task on the ICP) uses the STREAMS application interface to access STREAMS modules and drivers. STREAMS

modules and drivers run in the context of the calling task or the STREAMS scheduler task. Figure 1–2 shows how the standard Embedded STREAMS package functions in the OS/Impact environment.

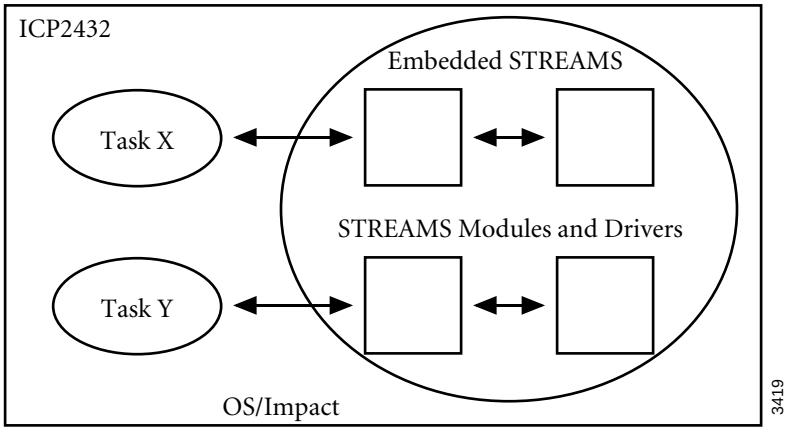

Figure 1–2: Embedded STREAMS Under OS/Impact

Embedded STREAMS does not provide a direct interface to the STREAMS sub-system from an application running on a host system. To provide such an interface, Simpact has augmented the Embedded STREAMS package with a STREAMS API for host applications and a corresponding OS/Impact STREAMS Manager ("Streaman") task. The host API and Streaman task communicate through the standard Simpact host device driver and XIO interface, using specially-defined message types and data formats. The API and Streaman essentially re-package STREAMS messages in the (non-STREAMS) format required by the host driver and XIO interface. Each stream opened by the host application corresponds to a single host driver/XIO connection (identified by node number). Figure 1–3 shows the complete Simpact STREAMS environment.

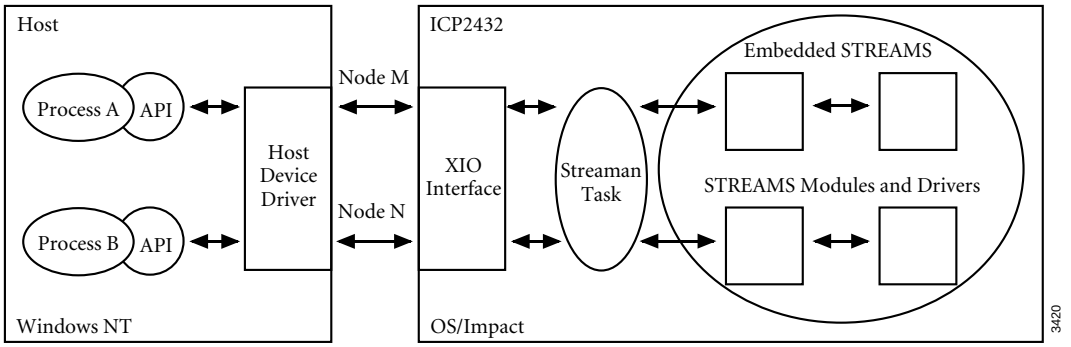

Figure 1–3: Simpact STREAMS Environment

Chapter

2 Application Interface to the Simpact STREAMS API

The Simpact STREAMS API provides applications on a Windows NT host access to Embedded STREAMS modules and drivers running on an ICP2432. The API provides only a blocking I/O interface to the ICP, and some functionality of Embedded STREAMS is not supported.

A global variable called `sserrno` is defined by the API and can be referenced by the application. When an API function fails, a general error code (-1 or -2) is returned and a specific error code is stored in `sserrno`.

When an API function fails due to an error on the host system (within the API, Windows NT system, or host device driver), the function returns the value -1. The error code stored in `sserrno` is one of the values defined in the Simpact header file `ssdefs.h` or in the Windows NT system header file `winerror.h`.

When an API function fails due to an error on the ICP2432 (within Embedded STREAMS or the Streaman task), the function returns the value -2. The error code stored in `sserrno` is one of the values defined in the Simpact header file `ssdefs.h` or in the Embedded STREAMS header file `eserrno.h`.

The following sections describe the functions provided by the API.

2.1 ss_open

The `ss_open` function is called to open a connection to a STREAMS driver on an ICP.

```
ss_open (
      char  *hostname,
      int   icpno,
      char  *device,
      int   stropt )
```

hostname is always `LOCAL`.

icpno is the ICP device number on which the stream is to be opened.

device is the device name string that identifies the STREAMS driver to be opened. Device name strings are defined in the `devices` array, in the Embedded STREAMS configuration file `uconfig.c`.

stropt has one of the following values, defined in the file `ssapi.h`:

SS_RDONLY Open stream for reading only

SS_WRONLY Open stream for writing only

SS_RDWR Open stream for reading and writing

If the value returned is greater than or equal to zero, the stream was successfully opened and the value is a session ID, used as an identifier for the newly opened stream. The session ID is provided on input to all remaining API calls to identify the stream on which the requested operation is to be performed. If the open request fails, the return value is less than zero, as follows:

-1 if the open request failed locally (within the host system) or

-2 if the open request failed remotely (on the ICP).

In either case, `sserrno` contains the specific error code.

2.2 ss_close

The ss_close function is called to close a STREAMS connection.

```
ss_close (
      int   sessID )
```

sessID identifies the connection to be closed.

The return value is zero if successful or one of the following values on failure:

-1 if the close request failed locally (within the host system) or

-2 if the close request failed remotely (on the ICP).

In either case, sserrno contains the specific error code.

2.3 ss_ioctl

The ss_ioctl function is called to perform a control (IOCTL) operation on a STREAMS connection.

```
ss_ioctl (
      int   sessID,
      int   command,
      int   arg )
```

sessID identifies the connection on which the control operation is to be performed.

command identifies the operation to be performed and has one of the following values, defined in the file ssdefs.h:

SS_IPUSH Push a module into the stream

SS_IPOP Remove the module at the top of the stream

SS_ILINK Link a stream under a multiplexing driver

SS_IUNLINK Unlink a stream from beneath a multiplexing driver

SS_ISTR Protocol-specific control request

SS_IFLUSH Flush read and/or write queues for the stream

arg is an optional argument associated with the IOCTL command.

The IOCTL commands and arguments are described in the sections that follow.

If the IOCTL request completes successfully, the return value is zero for all commands except `SS_ILINK`. For `SS_ILINK`, the return value is a mux ID that identifies the linked stream. (The mux ID is used on input to the `SS_IUNLINK` IOCTL request.) If the IOCTL request fails, the return value (for all commands) is one of the following:

-1 if the request failed locally (within the host system) or

-2 if the request failed remotely (on the ICP).

In either case, `sserrno` contains the specific error code.

2.3.1 SS_IPUSH

The `SS_IPUSH` command is used to push a module into the stream. The module is added just below the stream head. For this command, `arg` is a pointer to the name string of the module to be pushed. Module name strings are defined in the `fmodsw` array, in the Embedded STREAMS configuration file `uconfig.c`.

2.3.2 SS_IPOP

The `SS_IPOP` command pops a module from the stream. The module removed is always the one directly below the stream head. The `arg` parameter is not used.

2.3.3 SS_ILINK

The SS_ILINK command is used to link a stream underneath a multiplexing driver. The command is sent on a stream that is connected to the multiplexing driver and the arg parameter is the session ID of the stream to be linked under the driver.

If the request is successful, the return value is a mux ID that can be used later to unlink the stream (using the SS_IUNLINK IOCTL command). The multiplexing driver may also require the mux ID to be included in certain (protocol-specific) messages as a means of identifying the lower stream.

Once a stream has been linked under a multiplexing driver, no requests except ss_close can be sent on that connection until it has been unlinked.

2.3.4 SS_IUNLINK

The SS_IUNLINK command unlinks a stream from beneath a multiplexing driver. The command is sent on a stream that is connected to the multiplexing driver and the arg parameter is the mux ID of the stream to be unlinked. (The mux ID is returned from the SS_ILINK IOCTL request.)

2.3.5 SS_ISTR

The SS_ISTR command is a protocol-specific control request. For this command, arg is a pointer to an strioctl structure (defined in the file ssapi.h), which further defines the command format and operation.

```
struct strioctl
{
      int   ic_cmd;
      int   ic_timout;
      int   ic_len;
      char  *ic_dp;
};
```

ic_cmd is a protocol-defined command value.

ic_timout specifies the number of seconds the Streaman task should wait for a reply to the IOCTL request. (Streaman returns an error if the time-out period expires before a reply is received.)

ic_len is the number of bytes of data to be included in the request; on completion specifies the number of bytes of data returned.

ic_dp is a pointer to a data buffer, used to hold data passed into the request and also any data returned. The calling application must ensure that the data buffer is large enough to contain the returned data.

2.3.6 SS_IFLUSH

The `SS_IFLUSH` command can be used to flush read and/or write queues for a stream. For this command, `arg` specifies which queues are to be flushed and has one of following values (defined in `ssapi.h`):

FLUSHR Flush all read queues (including the API's internal read queue).

FLUSHW Flush all write queues.

FLUSHRW Flush all read and write queues (including the API's internal read queue).

2.4 ss_getmsg

The `ss_getmsg` function is called to read data from a streams connection. The first available message is always returned (specifying `RS_HIPRI` in the `flags` parameter on input is not supported).

```
ss_getmsg (
      int             sessID,
      struct strbuf   *ctlptr,
      struct strbuf   *dataptr,
      int             *flags )
```

sessID identifies the connection from which a message is to be read.

ctlptr is a pointer to a structure of type `strbuf` (defined below and in the file `ssapi.h`), used to describe the control part of the requested and returned message. If this parameter is `NULL`, the control part of the received message, if any, is discarded.

dataptr is a pointer to a structure of type `strbuf`, used to describe the data part of the message. If this parameter is `NULL`, the data part of the received message, if any, is discarded.

flags is a pointer to an integer location at which the API stores a value defining the priority of the returned message, as follows:

0	normal-priority message
SS_HIPRI	high-priority message

The `strbuf` structure, used to describe both the control and data parts of the returned message, is defined as follows:

```
struct strbuf
{
      int   maxlen;
      int   len;
      char  *buf;
};
```

For the `ctlptr` parameter, describing the control part:

maxlen is the number of bytes of data that can be accepted from the control part of the received message. `maxlen` is provided on input to the `ss_getmsg` request and is not used on return. If `maxlen` is -1, the control part of the received message, if any, is discarded.

len is the size of the control-part data actually returned and is not used on input to the request. If `len` is -1, the received message had no control part. If `len` is zero, the message had a control part of zero length.

buf is a pointer to a buffer in which the control-part data is to be stored on return from the request.

For the `dataptr` parameter, describing the data part:

maxlen is the number of bytes of data that can be accepted from the data part of the received message. `maxlen` is provided on input to the `ss_getmsg` request and is not used on return. If `maxlen` is -1, the data part of the received message, if any, is discarded.

len is the size of the data actually returned and is not used on input to the request. If `len` is -1, the received message had no data part. If `len` is zero, the message had a data part of zero length.

buf is a pointer to a buffer in which the data is to be stored on return from the request.

The return value from the `ss_getmsg` call is one of the following:

0 The request was successful.

-1 The request failed locally (within the host system).

-2 The request failed remotely (on the ICP).

SS_MORECTL The request may have been partially successful, but the caller did not provide sufficient buffer space for the entire control part of the received message (or the `ctlptr` parameter was `NULL`). The portion of the control-part data that could not be returned was discarded.

SS_MOREDATA The request may have been partially successful, but the caller did not provide sufficient buffer space for the entire data part of the received message (or the `dataptr` parameter was `NULL`). The portion of the data that could not be returned was discarded.

SS_MORECTL | SS_MOREDATA The request may have been partially successful, but the caller provided insufficient buffer space for both parts of the received message (or `NULL` `ctlptr` and/or `dataptr` parameters). The portions of the control and data parts that could not be returned were discarded.

If the request fails (return value is -1 or -2), `sserrno` contains the specific error code.

2.5 ss_putmsg

The `ss_putmsg` function is called to write data to a streams connection.

```
ss_putmsg (
      int             sessID,
      struct strbuf   *ctlptr,
      struct strbuf   *dataptr,
      int             flags )
```

sessID identifies the connection to which the message is to be written.

ctlptr is a pointer to a structure of type `strbuf` (defined below and in the file `ssapi.h`), used to describe the control part of the message. If this parameter is `NULL`, the message has no control part.

dataptr is a pointer to a structure of type `strbuf`, used to describe the data part of the message. If this parameter is `NULL`, the message has no data part.

flags defines the priority of the message, as follows:

0	normal-priority message
SS_HIPRI	high-priority message

The `flags` parameter is used only when the message has a control part (`ctlptr` is non-null and `ctlptr->len` is greater than or equal to zero).

The `strbuf` structure, used to describe both the control and data parts of the message, is defined as follows:

```
struct strbuf
{
      int   maxlen;
      int   len;
      char  *buf;
};
```

For the `ctlptr` parameter, describing the control part:

maxlen is not used.

len is the number of bytes in the control part of the message. If `len` is -1, the message has no control part. If `len` is zero, the message has a control part of zero length.

buf is a pointer to a buffer containing the control-part data.

For the `dataptr` parameter, describing the data part:

maxlen is not used.

len is the number of bytes in the data part of the message. If `len` is -1, the message has no data part. If `len` is zero, the message has a data part of zero length.

buf is a pointer to a buffer containing the data part of the message.

The caller's input parameters determine the type of STREAMS message created, as follows:

M_DATA Created when `ctlptr` is `NULL` or `ctlptr->len` is -1. (In this case, the `flags` parameter is not used.)

M_PROTO Created when `ctlptr` is non-NULL, `ctlptr->len` is greater than or equal to zero, and the `flags` parameter is zero.

M_PCPROTO Created when `ctlptr` is non-NULL, `ctlptr->len` is greater than or equal to zero, and `SS_HIPRI` is set in the `flags` parameter.

The return value is zero if successful or one of the following values on failure:

-1 if the `close` request failed locally (within the host system) or

-2 if the `close` request failed remotely (on the ICP).

In either case, `sserrno` contains the specific error code.

Chapter

3 Interface to the Simpact STREAMS Manager Task (Streaman)

This chapter describes the messages passed between the STREAMS API and the ICP's Streaman task.

Messages are generated by the API and passed to the Streaman task when applications call the API functions described in Chapter 2. These messages are internal to the API and Streaman task and are not used directly by Windows NT applications or by STREAMS modules and drivers resident on the ICP. You may find the information in this chapter useful if you want to increase your understanding of the internal operation of Simpact STREAMS or if you intend to modify the STREAMS API and/or the Streaman task.

3.1 Common Message Header

All messages exchanged between the API and the Streaman task begin with a common header, defined as follows:

```
typedef struct ss_hdr
{
    int     node;
    int     msgType;
    int     ctrlLen;
    int     dataLen;
    union
    {
        int         errno;
        struct ss_hdr *next;
    } u;
} SShdr;
```

node is the node number assigned to the connection. (For Windows NT, the node number is assigned by the host driver and can be obtained using the `IOCTL_ICP_GET_DRIVER_INFO` I/O control command.)

msgType identifies the format and function of the message using one of the values summarized in Table 3–1.

Table 3–1: Message Types

Message Type	Function	Direction of Transfer
`SS_OPEN`	Open a stream	to ICP
`SS_CLOSE`	Close a stream	to ICP
`SS_IOCTL`	IOCTL (device control) request	to ICP
`SS_PROTO`	Convey an `M_PROTO` message	to or from ICP
`SS_PCPROTO`	Convey an `M_PCPROTO` message	to or from ICP
`SS_DATA`	Convey an `M_DATA` message	to or from ICP
`SS_OPEN_ACK`	Reply to `SS_OPEN` request	from ICP
`SS_CLOSE_ACK`	Reply to `SS_CLOSE` request	from ICP
`SS_IOC_ACK`	Reply to `SS_IOCTL` request	from ICP
`SS_ERROR`	Fatal error indication	from ICP

ctrlLen specifies the number of bytes in the control part of the message and is zero if there is no control part. If `ctrlLen` is non-zero, the control part of the message immediately follows the message header.

dataLen specifies the number of bytes in the data part of the message and is zero if there is no data part. If `dataLen` is non-zero, the data part of the message immediately follows the control part (or immediately follows the message header if `ctrlLen` is zero).

errno contains a completion code for replies (`SS_OPEN_ACK`, `SS_CLOSE_ACK`, and `SS_IOC_ACK`) and contains an error code for `SS_ERROR`. For a successful

completion, errno is zero. For error completions or for SS_ERROR, errno contains a standard STREAMS error code from the file eserrno.h.

next is used only while a received SS_DATA, SS_PROTO, or SS_PCPROTO message is queued internally to the STREAMS API and contains a pointer to the header of the next message on the queue.

Note

There is a special "abort" command (DLI_ICP_CMD_TERM) sent from the Windows NT host driver when an application terminates without closing its connection to the driver. The DLI_ICP_CMD_TERM command code, contained in a special structure of type APIHDR, overlays the high-order word of the ssHdr structure's msgType field. To allow easy identification of this command, keep the high-order word of all message types equal to zero. In addition, beware of changing the format of the ssHdr structure. (See note in file smdefs.h.)

3.2 Format of the SS_OPEN Request

For an SS_OPEN request, the fields of the message header are used as follows:

node identifies the connection.

msgType is SS_OPEN.

ctrlLen is the size of the SSopen structure defined below.

dataLen is the length of the device name string (including NULL terminator).

errno is not used.

The control part of the message immediately follows the message header and is defined by the following structure:

```
typedef struct
{
    int option;
} SSopen;
```

option has one of the following values, defined in the Embedded STREAMS header file `esfcntl.h`:

O_RDONLY Open stream for reading only

O_WRONLY Open stream for writing only

O_RDWR Open stream for reading and writing

The data part of the message immediately follows the `SSopen` structure and consists of a null-terminated name string identifying the device (STREAMS driver) to be opened. The specified string must match a device name string from the `devices` array in the Embedded STREAMS configuration file `uconfig.c`.

3.3 Format of the SS_OPEN_ACK Reply

For an `SS_OPEN_ACK` reply, the fields of the message header are used as follows:

node identifies the connection.

msgType is `SS_OPEN_ACK`.

ctrlLen is the size of the `SSopack` structure defined below.

dataLen is zero.

errno is zero if successful or contains an error code on failure.

The control part of the message immediately follows the message header and is defined by the following structure.

```
typedef struct
{
    int strfd;
} SSopack;
```

strfd is the file descriptor assigned to the stream. The file descriptor is used by the Streaman task on the ICP to identify a stream. The API saves the file descriptor so it can be supplied to Streaman on the SS_ILINK IOCTL request (see Section 3.6 on page 34), but uses it for no other purpose.

This message has no data part.

3.4 Format of the SS_CLOSE Request

For an SS_CLOSE request, the fields of the message header are used as follows:

node identifies the connection to be closed.

msgType is SS_CLOSE.

ctrlLen is zero.

dataLen is zero.

errno is not used.

This message has no control part and no data part.

3.5 Format of the SS_CLOSE_ACK Reply

For an SS_CLOSE_ACK reply, the fields of the message header are used as follows:

node identifies the connection.

msgType is SS_CLOSE_ACK.

ctrlLen is zero.

dataLen is zero.

errno is zero if successful or contains an error code on failure.

This message has no control part and no data part.

3.6 Format of the SS_IOCTL Request

For an SS_IOCTL request, the fields of the message header are used as follows:

node identifies the connection.

msgType is SS_IOCTL.

ctrlLen is the size of the SSioctl structure defined below.

dataLen specifies the number of bytes of data associated with the request (see Table 3–2).

errno is not used.

The control part of the message immediately follows the message header and is defined by the following structure:

```
typedef struct
{
    int command;
    int argument;
    int timeout;
} SSioctl;
```

command is one of the following IOCTL command codes, defined in the file `ssdefs.h`:

SS_IPUSH	Push a module into the stream
SS_IPOP	Remove the module at the top of the stream
SS_ILINK	Link a stream under a multiplexing driver
SS_IUNLINK	Unlink a stream from beneath a multiplexing driver
SS_ISTR	Protocol-specific control request
SS_IFLUSH	Flush read and/or write queues for the stream

argument is an optional argument, depending on the command type.

timeout is used only for command type `SS_ISTR` and specifies the number of seconds to wait for a reply to the request. This value is taken directly from the `ic_timout` field of the `strioctl` structure supplied by the application on input to the `ss_ioctl` API function.

The data part of the message, if any, immediately follows the `SSioctl` structure.

For the `SS_IOCTL` request, the uses of the `dataLen` field of the message header and the `argument` field of the `SSioctl` structure, as well as the contents of the data part of the message, depend on the IOCTL command type, as described in Table 3–2.

Table 3–2: IOCTL Request Formats

Command Type	argument	dataLen	Data Contents
SS_IPUSH	Not used	Length of the module name string (including NULL terminator)	The name of the module to be pushed into the stream (from the fmodsw array in the Embedded STREAMS configuration file uconfig.c)
SS_IPOP	Not used	zero	none
SS_ILINK	The file descriptor of the stream to be linked (see Section 3.3 on page 32)	zero	none
SS_IUNLINK	The Mux ID identifying the stream to be unlinked (see Section 3.7 on page 37)	zero	none
SS_ISTR	The value specified in the ic_cmd field of the strioctl structure on input to the ss_ioctl API function	The value specified in the ic_len field of the strioctl structure	Copied from the data buffer (ic_dp) supplied on input to ss_ioctl
SS_IFLUSH	Flush type flag, defined in the file esstropt.h: FLUSHR to flush read queues, FLUSHW to flush write queues, or FLUSHRW to flush read and write queues	zero	none

3.7 Format of the SS_IOC_ACK Reply

For an SS_IOC_ACK reply, the fields of the message header are used as follows:

node identifies the connection.

msgType is SS_IOC_ACK.

ctrlLen is zero.

dataLen specifies the number of data bytes returned (see Table 3–3).

errno is zero if successful or contains an error code on failure.

This message has no control part. The data part, if any, immediately follows the message header. The data length and contents depend on the IOCTL command type, as described in Table 3–3.

Table 3–3: IOCTL Reply Formats

Command Type	dataLen	Data Contents
SS_IPUSH	zero	none
SS_IPOP	zero	none
SS_ILINK	4	Mux ID identifying the linked stream (used on input to the SS_IUNLINK IOCTL request)
SS_IUNLINK	zero	none
SS_ISTR	Implementation-dependent	Copied to the data buffer (ic_dp) supplied on input to ss_ioctl
SS_IFLUSH	zero	none

3.8 Format of the SS_PROTO and SS_PCPROTO Messages

For an SS_PROTO or SS_PCPROTO message, the fields of the message header are used as follows:

node identifies the connection.

msgType is SS_PROTO or SS_PCPROTO.

ctrlLen is the length of the control part of the message or -1 if there is no control part.

dataLen is the length of the data part of the message or -1 if there is no data part.

errno is not used.

For an SS_PROTO or SS_PCPROTO message traveling from the host to the ICP, the control and data parts of the message are supplied on input to the API function ss_putmsg. For a message traveling from the ICP to the host, the control and data parts are as supplied in the corresponding M_PROTO or M_PCPROTO obtained from the stream head.

The control part, if any, immediately follows the message header. The data part, if any, immediately follows the control part. If there is no control part or if the length of the control part is zero, the data part immediately follows the message header.

3.9 Format of the SS_DATA Message

For an SS_DATA message, the fields of the message header are used as follows:

node identifies the connection.

msgType is SS_DATA.

ctrlLen is zero.

dataLen is the length of the data part of the message.

errno is not used.

For an SS_DATA message traveling from the host to the ICP, the data part of the message is supplied on input to the API function ss_putmsg. For a message traveling from the ICP to the host, the data part is as supplied in the corresponding M_DATA message obtained from the stream head.

This message type has no control part. The data part of the message immediately follows the message header.

3.10 Format of the SS_ERROR Message

For an SS_ERROR message, the fields of the message header are used as follows:

node identifies the connection.

msgType is SS_ERROR.

ctrlLen is zero.

dataLen is zero.

errno contains an error code.

This message has no control part and no data part.

Chapter

4 Simpact STREAMS Development Environment

Some portions of the Simpact STREAMS software are distributed in object format only, while others are distributed in source form. The two main components are the client-resident software, including the API and sample client application, and the ICP-resident software, including the Simpact STREAMS system-level software and STREAMS Manager task.

4.1 Installation

Before installing the Simpact STREAMS software, install the Simpact toolkit and embedded software packages as described in the *ICP2432 User's Guide for Windows NT*. Simpact STREAMS is shipped on separate media, but is installed under the `freeway` directory created for the toolkit and embedded software.

To install Simpact STREAMS, insert the release media, run `setup.exe`, and follow the instructions displayed on the screen.

4.2 Directory Structure

The `freeway/client/test/streams` directory contains the client-resident Simpact STREAMS API in source form and a sample client program that uses the API to communicate with the ICP-resident sample loopback driver and filter module.

Under the `freeway/icpcode/streams` directory, the following sub-directories provide the server-resident software:

lib object libraries containing the Simpact STREAMS system-level software:

- **esLib.a** STREAMS system services
- **esULib.a** STREAMS configuration and Streaman task
- **supdvLib.a** miscellaneous drivers
- **tbLib.a** timebase driver

include header files containing STREAMS constants and structure definitions

config STREAMS configuration files, in source form

streaman STREAMS manager task, in source form

sample sample STREAMS driver, module, and multiplexing driver for use with the sample client program

dvr miscellaneous sample STREAMS drivers and modules, in source form

Several header files in the `freeway/icpcode/streams` directory are included by the sample client program. These are:

- `ssif.h` and `ssdefs.h` in the `freeway/icpcode/streams/streaman` directory and
- `ssloop.h` and `ssmux.h` in the `freeway/icpcode/streams/sample` directory.

4.3 ICP-Resident Executable Modules

In the OS/Impact environment, separate executable software modules can be built and independently loaded to the ICP. Application modules communicate with the OS/Impact system services module by making system calls, which execute software trap instructions and do not require the calling module to know the address of the ultimate system function.

This concept is not followed within the Embedded STREAMS environment. STREAMS applications make direct subroutine calls to the stream head, and STREAMS modules and drivers make direct subroutine calls to access STREAMS services. Because of this, all software that uses STREAMS resources, including the Streaman task, must be linked into a single module together with the system-level STREAMS code itself.

In the Simpact STREAMS environment, only two modules are generally loaded to the ICP: the OS/Impact system services module and a STREAMS application module. Additional modules could be loaded, but these modules could access only the conventional OS/Impact services; they would have no access to STREAMS services.

4.4 STREAMS Configuration

Configuration files for the standard system portion of Embedded STREAMS reside in the `freeway/icpcode/streams/config/osimpact/all` directory. Two files in this directory are intended to be modified to suit your STREAMS requirements. These are `usrparam.h` and `uconfig.c`. The other files in this directory contain OS/Impact-specific or ICP2432 target-specific code, structures, and constant definitions, and should not be modified.

The file `uconfig.c` allocates and initializes the Embedded STREAMS configuration structures and `usrparam.h` contains constant definitions used to initialize some of those structures. The structure formats are defined in the header files `esswitch.h` and `esinit.h` (in the `freeway/icpcode/streams/include` directory).

The Streaman task can be configured to suit your requirements by modifying the header file `smdefs.h` in the `freeway/icpcode/streams/streaman` directory.

4.4.1 Modifying the File usrparam.h

For descriptions of the tuneable parameter values in `usrparam.h`, refer to the manual page for the administrative function `es_init` in the document *Embedded STREAMS Programmer's Guide & Reference*. For the OS/Impact implementation of Embedded

STREAMS, several parameters differ from those described in the standard documentation. The `ES_STACK_SIZE` parameter is deleted and the following parameters are added:

ES_TASK_ID Task ID for the STREAMS scheduler task

ES_STACK_ADDR Initial stack address for the STREAMS scheduler task

ES_SLICE Time slice enable/disable flag for the STREAMS scheduler task (1=enable, 0=disable)

NPRIORITY Number of scheduling priorities (see Section 4.4.2)

Several additional parameters in the file `usrparam.h` have been added for OS/Impact support. These parameters are used in the file `uconfig.c` to allocate space for:

- an array of extended task control blocks (Embedded STREAMS requires one such control block for each task that accesses STREAMS) and
- the STREAMS workspace, which is used to build the STREAMS data structures, including message blocks, data blocks, and data buffers.

The parameters are as follows:

XTSIZE This parameter should not be modified. It configures the size of the extended task control block structure, defined in the file `sysparam.h`.

MAXTASKS This parameter configures the maximum number of OS/Impact tasks that will access STREAMS. In general, only the Streaman task and its auxiliary Pollman task access STREAMS. However, for simplicity, this parameter can be set to the same value as the `cf_ntsk` field of the OS/Impact configuration table.

WORK_SIZE This parameter defines the size of the STREAMS workspace in longwords. The size required depends on the values you have configured for various tuneable parameters. If the configured workspace size is too small, the STREAMS system will panic during its initialization.

Note

A call to the `es_init` function is generated automatically the first time an application task (the Streaman task) opens a stream. It is not necessary to call `es_init` directly.

4.4.2 STREAMS Extensions

Simpact has enhanced Embedded STREAMS with the additional feature of prioritized scheduling. The programmer assigns a scheduling priority to each STREAMS module using an added field in the `module_info` structure. The number of scheduling priorities is configured using the tuneable parameter `NPRIORITY` as defined in Section 4.4.1.

Using the `mi_pri` field of the `module_info` structure, each module is assigned a scheduling priority from `0` through `NPRIORITY - 1`, where `0` is the highest priority. All queues associated with a particular module execute at the same priority.

For each configured priority, the system maintains a list of queues waiting for execution at that priority. When a queue is scheduled for execution, it is added to the appropriate list based on the priority specified in the associated module's `module_info` structure. When the STREAMS scheduler is ready to dispatch a queue for execution, it searches the scheduling lists in order of priority and chooses the longest-waiting queue on the first non-empty list.

If you do not want to use prioritized scheduling, set `NPRIORITY` to `1` and set the `mi_pri` field of the `module_info` structure to `0` for all modules.

4.4.3 Modifying the File uconfig.c

In the file `uconfig.h`, there are several arrays of structures that you will need to modify in order to add STREAMS modules and drivers to the system.

For each STREAMS driver, add an entry to the `cdevsw` array. The position in the array determines the driver's major device number. Above the `cdevsw` array, be sure to extern the address of the driver's `streamtab` structure.

For each STREAMS module, add an entry to the `fmodsw` array. The name string you enter into this array must match the module name specified by an application when it pushes the module into the stream. As noted above, be sure to add an extern for the module's `streamtab` structure.

In the `devices` array, add one or more entries for each of your STREAMS drivers. Each entry creates a minor device for the driver. For a standard device, specify your driver's major device number (based on its index in the `cdevsw` array) and a minor device number. For a cloneable device, specify the clone device's major number (based on the index of the "clone" device in the `cdevsw` array) and the major number of your driver. (A single driver can have both standard and cloneable entries in the `devices` array.)

Note

If you do not plan to use the sample drivers and modules provided with Simpact STREAMS, their entries can be removed from the `cdevsw`, `fmodsw`, and `devices` arrays. (If you have any cloneable drivers, do not remove the "clone" entry from the `cdevsw` array.)

4.4.4 Modifying the File smdefs.h

The Streaman task can be configured by modifying certain parameters in the header file `smdefs.h` in the `freeway/icpcode/streams/streaman` directory. The following parameters can be modified:

MAX_SSMSG_SIZE Maximum STREAMS message size, including both control and data parts

MAX_SSCTL_SIZE Maximum size of the control part of a STREAMS message

MAX_SSDATA_SIZE	Maximum size of the data part of a STREAMS message
MAX_SESSIONS	Maximum number of streams supported (cannot be greater than 124)
CLOSE_TIME	The number of seconds Streaman will delay to wait for downstream data to drain before closing a stream
DATA_BASE	Base address of the data buffer partition used for host communication
DATA_TOP	Ending address of the data buffer partition
REQ_BASE	Base address of the host request buffer partition used for host communication
REQ_TOP	Ending address of the host request buffer partition

4.4.5 OS/Impact Configuration

OS/Impact configuration is performed separately from STREAMS configuration and is described in the *Freeway OS/Impact Programmer's Guide*. Your executable module must include a system initialization procedure similar to the function `sysinit` in the file `sample.asm` (in the `freeway/icpcode/streams/sample` directory). The system initialization procedure jumps to OS/Impact's initialization procedure, passing it the address of a configuration table and a list of task initialization structures.

The fields of the OS/Impact configuration table are listed below together with comments describing factors you should consider when using Embedded STREAMS:

cf_ntsk Three OS/Impact tasks are required to support Embedded STREAMS. These are the STREAMS scheduler task, the Streaman task, and Streaman's auxiliary Pollman task. Task IDs 3, 4 and 5, respectively, are assigned to these three tasks. Any number of STREAMS modules and

drivers can be supported without requiring any additional OS/Impact tasks.

cf_npri The STREAMS scheduler must be higher priority than any task that accesses STREAMS, so `cf_npri` must be set to at least 2. As shipped, the priority of the scheduler is 2 (configured using the tuneable parameter `ES_PRIORITY`) and the sample `sysinit` function initializes the Streaman task at priority 3 and the Pollman task at priority 4 (requiring a `cf_npri` value of at least 4).

cf_nque The number of queues required by Embedded STREAMS depends on the number of streams supported within the Streaman task, as defined by the `MAX_SESSIONS` parameter in the file `smdefs.h`. Streaman uses queue IDs 5 through n, where n is 8 + `MAX_SESSIONS` x 2.

cf_nlrm The number of alarms required by Embedded STREAMS also depends on the number of streams supported. Streaman requires one alarm for each stream and uses alarm IDs 1 through n, where n is equal to `MAX_SESSIONS`.

cf_npar The Streaman task creates two partitions, using partition IDs 1 and 2. These partitions occupy memory in the ranges defined by the parameters `DATA_BASE`, `DATA_TOP`, `REQ_BASE`, and `REQ_TOP` in the file `smdefs.h`.

cf_nrsc Embedded STREAMS does not use resources.

cf_ltik The STREAMS tuneable parameter `TICKS_PER_SEC` should correspond to the value defined for the `cf_ltik` parameter of the OS/Impact configuration table. For example, if `cf_ltik` is set to 100 (100 milliseconds per tick), `TICKS_PER_SEC` should be set to 10 (10 ticks per second).

cf_ltsl This parameter can be adjusted as needed for system performance.

cf_cisr Embedded STREAMS does not require a user clock interrupt service routine.

Following the configuration table, you must supply task initialization structures for the Streaman task and its auxiliary Pollman task. The structures should be initialized as shown in the sample file `sample.asm`, although you may relocate the initial stack addresses if needed.

4.4.6 Modifying the Memory Map

A number of elements in the OS/Impact system reside at fixed (hard-coded) addresses. When modifying any of these addresses, conflicts with other system elements must be avoided. The system elements to be considered are summarized below:

- The system services (OS/Impact) module must be loaded at address 0x801200. (This load address cannot be modified.)
- The `makefile` in the `freeway/icpcode/streams/sample` directory specifies a load address of 0x818000 for the Embedded STREAMS executable module.
- The file `smdefs.h` in `freeway/icpcode/streams/streaman` assigns Streaman's data buffer partition to address range 0x890000 - 0x8D0000 and its host request buffer partition to address range 0x8D0000 - 0x8F0000.
- The file `sample.asm` in `freeway/icpcode/streams/sample` sets the initial stack pointer for the Streaman task to 0x8F8000 and the stack pointer for the Pollman task to 0x8F4000. Since stacks grow from higher to lower memory addresses, and allowing 0x1000 bytes of stack space for each task, the address ranges 0x8F3000 - 0x8F4000 and 0x8F7000 - 0x8F8000 should be reserved for these stacks.
- The file `usrparam.h` in `freeway/icpcode/streams/config/osimpact/all` (parameter `ES_STACK_ADDR`) sets the initial stack pointer for the STREAMS scheduler task to address 0x8F7000. Again allowing 0x1000 bytes, address range 0x8F6000 - 0x8F7000 should be reserved for the scheduler's stack.

4.5 Building the Embedded STREAMS Software

After modifying configuration parameters as needed, the Embedded STREAMS user library (`esULib.a`) can be rebuilt by running "make esconfig" in the `freeway/icpcode/streams/config/osimpact/icp2432` directory. The user library contains the configurable portion of Embedded STREAMS as well as the Streaman task. The system library, `esLib.a`, is provided in object format only and cannot be rebuilt.

After rebuilding the user library, you must link your STREAMS modules and drivers with the user and system libraries. If your configuration (`uconfig.c`) references the sample modules and drivers contained in the libraries `supdvLib.a` and `tbLib.a`, you'll need to link with those libraries as well. The `makefile` provided in the `freeway/icpcode/streams/sample` directory can be used as a model for this procedure.

4.6 STREAMS Initialization

After building your executable module, you must download it and the system services module to the ICP as described in the *Freeway Embedded ICP2432 User's Guide for Windows NT*. (The name of the download script is `streams.nt`.) Execution begins at your module's system initialization entry point. Your module provides task initialization structures to OS/Impact that allow it to create and dispatch the Streaman and Pollman tasks.

Embedded STREAMS is not initialized, and the STREAMS scheduler task is not created, until the first time a STREAMS service is accessed. Unless you have created other OS/Impact tasks that use STREAMS services, STREAMS initialization does not occur until a host application opens a connection to a STREAMS driver on the ICP using the `ss_open` API call.

When the Streaman task passes the first open request to Embedded STREAMS (using the `es_open` call), the system detects that STREAMS has not been initialized and creates the STREAMS scheduler task. When the scheduler task is dispatched, its first action is

to call the function `es_init`. (This function is described in the Administrative Interface section of the *Embedded STREAMS Programmer's Guide & Reference* manual.) The address of the `es_config` structure (defined in the configuration file `uconfig.c`) is passed on input to `es_init` and provides pointers to the tuneable parameters and other structures used to initialize the STREAMS system and build the STREAMS message blocks, data blocks, and other data structures.

Index

W

X–Z

Customer Report Form

We are constantly improving our products. If you have suggestions or problems you would like to report regarding the hardware, software or documentation, please complete this form and mail it to Simpact at 9210 Sky Park Court, San Diego, CA 92123, or fax it to (619) 560-2838.

If you are reporting errors in the documentation, please enter the section and page number.

Your Name: _____

Company: _____

Address: _____

_____

_____

Phone Number: _____

Product: _____

Problem or
Suggestion: _____

_____

_____

_____

_____

_____

Simpact, Inc.
Customer Service
9210 Sky Park Court
San Diego, CA 92123