

UNCLASSIFIED ↓


REVISION HISTORY (U)			
REV	DESCRIPTION (U)	DATE	APPROVED
-	Initial Release	8/10/04	L. Freitag

Compact Data Layer for Acoustic Communications

NEXT ASSY	USED ON

THE REVISION STATUS OF ALL SHEETS OF THIS DRAWING IS THE SAME AS SHEET 1

APPLICATION



CONTRACT NO	
DESIGN Freitag/Singh	DATE 7/2004
APVD L. Freitag	DATE 8/10/04
CHECKED	
APVD	
APVD	
APVD	
APVD	

Woods Hole Oceanographic Institution 266 Woods Hole Road Woods Hole, MA 02540		
401002-SPEC Compact Data Layer for Acoustic Communications		
SIZE A	CAGE CODE 88846	DWG NO 401002-SPEC
SCALE NONE		SHEET 1 of 10

UNCLASSIFIED ↑

Table of Contents

1	SCOPE	3
1.1	INTRODUCTION	3
2	DATA FRAME STRUCTURE	3
2.1	SEGMENTER	3
2.2	CRC.....	4
2.3	HEADER STRUCTURE	4
2.4	SOURCE AND DESTINATION	4
2.5	FRAME NUMBER	4
2.6	ACKNOWLEDGEMENT BIT	4
2.7	FRAME FULL BIT	5
2.8	DATA WHITENING	5
3	SOURCE CODE FOR DATA LAYER	6
3.1	SOURCE CODE INFORMATION	6
3.1.1	<i>Example of Data-Packet Generation Using Micro-Modem Standard Data Layer (sdl.h)</i>	6
3.1.2	<i>Example of Data Frame Generation (sdlo.c)</i>	6
3.1.3	<i>Example of Making Data Frame Header (sdlfun.c)</i>	7
3.1.4	<i>Make Frame (sdlfun.c)</i>	8
3.1.5	<i>Check Frame Check Sequence (sdlfun.c)</i>	9
3.1.6	<i>16-bit CRC function (sdlfun.c)</i>	9
3.1.7	<i>16-bit CRC table (sdlfun.c)</i>	10

Table of Figures

Figure 1	Data Frame Structure	3
Figure 2	Structure of Frame Header (16 bits)	4

SIZE	CAGE CODE	DWG NO	REV
A	88846	401002-SPEC	
SCALE		SHEET	2

Document Overview

This document is based in part upon a previous WHOI Document *Multi-User Frequency Hopping Underwater Acoustic Communication Protocol* version 1.02 dated May 25, 2000. This new document, designated 401002-SPEC replaces section 2 of that document in its entirety with the exception of coding and interleaving.

Section 1: Scope and Introduction

Section 2: Data Layer Description

Section 3: Source Code Examples for Data Layer

1 Scope

This document describes the precise format of the data layer used on the WHOI Micro-Modem and the Utility Acoustic Modem. The intent of this document is to allow replication of WHOI standard frames and packets at a bit level.

1.1 Introduction

The data layer used by WHOI for acoustic communications includes the following components:

1. Segmentation of data into frames.
2. Addition of header information for each frame
3. Data integrity check for each frame using a CRC.

This document does not include any details of modulation and modulation-specific signaling such as training sequences or timing acquisition.

2 Data Frame Structure

Data packets are made up of a number of frames. Each frame consists of three sections:

1. Cyclic Redundancy Check (CRC). 16 bits (2 bytes).
2. Header. 16 bits (2 bytes).
3. Data Payload, 32, 64, 128, or 256 bytes.

See functions in section 3 for additional details on packing and unpacking these components.

Data Frame Structure				
0	1	2	3	4
CRC		Header		Data Payload

Figure 1 Data Frame Structure

2.1 Segmenter

The segmenter works with data units of maximum size 16128 bytes. Each unit is broken up into frames of length n , where n can take on the following discrete values: 256, 128, 64 or 32 bytes based on the data rate chosen. A 2-byte header and a 2-byte Frame Check Sequence (FCS) are added to each frame. Figure 1 shows an overview of the frame structure. A maximum of 63 frames can be concatenated together to form a single packet for error-correction coding and modulation. If a frame consists of less than n bytes of data, it is zero-padded to the defined frame size and a

SIZE	CAGE CODE	DWG NO	REV
A	88846	401002-SPEC	
SCALE		SHEET	3

header bit is accordingly set.

2.2 CRC

A 16-bit cyclic redundancy-check (CRC) is appended to each frame and used as a Frame Check Sequence. The CRC is the CCITT standard, with a generator polynomial $g(x)=x^{16}+x^{12}+x^5+1$, and a minimum distance ($d=4$) over block lengths of up to 32752 symbols (Wicker, 1995. *Error Control Systems for Digital Communication and Storage*.) The CRC protects both the data and the header.

2.3 Header Structure

The header of each frame is 2 bytes and is constructed as follows.

Header Bit Definitions					
Name	Full	ACK	Frame Number	Destination Address	Source Address
# Bits	1	1	6	4	4
Position	0	1	2-7	8-11	12-15

Figure 2 Structure of Frame Header (16 bits)

2.4 Source and Destination

Bits 8 to 11 are assigned as destination address, while bits 12 to 15 are assigned for source addressing. The address at the physical layer is intended to allow the data layer and those layers which are higher (e.g. network layer) to accept, reject, or route, data frames received at a particular frequency, data rate and coding method. The 0000 destination is typically reserved for the base node in a system, but that is optional. While a network may consist of many more nodes than 16, at the physical layer only 16 will be configured such that they will receive each other's packets.

2.5 Frame Number

Bits 2 through 7 are reserved for frame numbers. Frame number 0 is interpreted as a supervisory frame and numbers 1 through 63 are interpreted as data frames. Frame number always starts at 1 and numbers the frames within a single packet.

Supervisory frames are not defined in this document. They were originally defined in the FH-FSK specification.

2.6 Acknowledgement Bit

Bit 1 is the Acknowledgement (ACK). If this bit is set, an ACK is requested when the receiver correctly decodes the frame. The definition of the response to a frame received with the ACK bit set is system dependent. The methodology for handling the ACK in the Micro-Modem is described in the Micro-Modem Software Interface Guide Version 2.7 and higher.

SIZE A	CAGE CODE 88846	DWG NO 401002-SPEC	REV -
SCALE		SHEET 4	

2.7 Frame Full Bit

Bit 0, if set, signifies that the frame is completely full of data destined for the data layer. If not set, it indicates that the frame is padded to make it the standard size. If this bit is set then byte number 5 contains an eight bit number which indicates how many bytes of actual data are present in the frame. This allows the segmenter to deliver only exactly the same number of bytes at the destination as were sent at the source.

2.8 Data Whitening

The data is whitened using an m -sequence which is generated using a shift register of length 11 and a generator polynomial $g(x) = 1+x^2 + x^{11}$. The shift register is initialized with a one and 10 zeros. The whitening sequence generated in this fashion is XOR'd with the data prior to coding or interleaving. Details of coding and interleaving are dependent upon the ECC layer of the transmission system and not described here.

SIZE	CAGE CODE	DWG NO	REV
A	88846	401002-SPEC	-
SCALE		SHEET	5

3 Source Code for Data Layer

Source code to generate and check a data frame is provided below. The source code is for illustrative purposes only. Actual source code should be requested from WHOI and checked out from the source code control environment. All of the following source code is from the WHOI Micro-Modem, which is a 16-bit integer processor (TI C5416).

3.1 Source Code Information

The source code described here is for a single frame only. Multiple-frame encoding is not included. The source code is from `sdl.h`, `sdl.c`, `sdlo.c`, and `sdlfun.c`.

3.1.1 Example of Data-Packet Generation Using Micro-Modem Standard Data Layer (`sdl.h`)

The generation of one or more data frames is done using information maintained on the Micro-Modem in the Standard Data Layer (SDL) structure: `SDL_Obj`.

```
typedef struct {
    int srcid;          // 2-15 ( 0,1 reserved for base, broadcast )
    int destid;        // 0-15
    int frametype;     // 0: supervisory 1-63, data
    int ack;           // ack or not
    int framesize;     // frame size in 8-bit bytes
    int pkttype;       // chooses the ecc
    void *ibuf;        // points to rdframe output buffer: 2 * datasize
    void *obuf;        // points to mkframe output buffer: 2 * framesize
    int oframes;       // counts number of frames created
    int iframes;       // counts number of frames read
    int mode;          // 4 bits
    int ctype;         // 3 bits
    int grp;           // 4 bits
    int rate;          // 4 bits
    int arq;           // 4 bits, repeat request list
} SDL_Obj;
```

3.1.2 Example of Data Frame Generation (`sdlo.c`)

Making a single data frame is done using `mkframe` with a call such as that shown here using elements of the `SDL_obj`.

```
mkframe(s->srcid,
        s->destid,
        s->frametype,
        s->ack,
        inbuf->data,
        inbuf->len,
        s->framesize,
        outbuf->data);

s->oframes++;
```

SIZE A	CAGE CODE 88846	DWG NO 401002-SPEC	REV -
SCALE		SHEET 6	

3.1.3 Example of Making Data Frame Header (sdlfun.c)

The function mkheader creates the 16 bit packed header.

```
/* Make 16 bit header */  
  
int mkheader(int src,int dest,int framenum,int ackrequest,int full){  
    int header ;  
  
    header = ( (src & 0x0f) << 12) |  
             ( (dest & 0x0f) << 8) |  
             ( (framenum & 0x03f) << 2) |  
             ( (ackrequest & 0x01)<< 1) |  
             ( (full & 0x0001) ) ;  
    return(header);  
}
```

SIZE A	CAGE CODE 88846	DWG NO 401002-SPEC	REV -
SCALE		SHEET 7	

3.1.4 Make Frame (sdlfun.c)

```
/* make a single frame from single frames worth of data, header info
*/
void mkframe(int src,           // source address
             int dest,         // destination address
             int ftype,        // frame type (0:sup, 1..nframes:data)
             int ack,          // ack or not
             unsigned int *data, // points to unpacked data bytes
             int datasize,     // number of bytes in *data
             int framesize,    // size of output frame in bytes (packed % 2)
             unsigned int *frame) // pointer to output data
{
    int l, fwords, nf, sf, full;
    unsigned int dbyte;

    fwords = framesize / 2 ;           // 2 bytes per word
    memset(frame, 0, (fwords*sizeof(unsigned int))); // fill with zeros

    /* now fill frame */
    if (datasize==(framesize-4)) { // if frame is full
        nf=2;                       // start with f[2]
        sf=8;                         // start with upper byte
        full=1;
    } else {
        nf=2;                       // start with f[2]
        sf=0;                         // start with lower byte
        frame[nf]|=(datasize&(0x0fff))<<8; // fill in the datasize
        full=0;
    }

    /* first fill data */
    for (l=0;l<datasize;l++){
        dbyte= data[l] & (0x0ff) ; // input vector NOT packed
        frame[nf] |= (dbyte << sf) ; // pack the byte into the frame

        /* increment indices */
        if((sf--=8)<0){ // decrement shift, if neg
            sf=8; // reset shift to upper byte
            nf++; // of next frame word
        }
    }

    /* now put in the header */
    frame[1]=(0x0ffff) & mkheader(src, dest, ftype, ack, full) ;

    /* now the CRC */
    frame[0] |= crc(frame, fwords, 1);
}

```

SIZE A	CAGE CODE 88846	DWG NO 401002-SPEC	REV -
SCALE		SHEET 8	

3.1.5 Check Frame Check Sequence (sdlfun.c)

Check the CRC of a frame.

```
int ckpkt(int *indat)
{
    unsigned int sentfcs, recvfcs;

    sentfcs = indat[0];
    indat[0] = 0;
    recvfcs = crc(indat, 18, 1);
    indat[0] = sentfcs;

    return (sentfcs==recvfcs);
}
```

3.1.6 16-bit CRC function (sdlfun.c)

```
unsigned int crc(unsigned int *vec,int length,int comp){

    int fcs=PPPINITFCS ;
    int c ;

    c=length;

    while(c--) { /* loop through 16 bit words */
        /* byte by byte - MSB -> LSB */
        fcs = ((fcs & 0x0ff00) >> 8) ^ fcstab[ ( ((fcs & 0xffff) ^ ((*vec & 0xff00)
>> 8)) ) & 0xff];
        fcs = ((fcs & 0x0ff00) >> 8) ^ fcstab[ ( ((fcs & 0xffff) ^ ((*vec++ &
0x00ff))) ) & 0xff];
    }
    fcs=fcs&0xffff;
    if (comp){
        fcs=(~fcs)&0xffff);
    }
    return(fcs);
}
```

SIZE A	CAGE CODE 88846	DWG NO 401002-SPEC	REV -
SCALE		SHEET 9	

3.1.7 16-bit CRC table (sdlfun.c)

```

/* CRC-16 bit table for table lookup */
static int fcstab[256] = {
    0x0000,0x1189,0x2312,0x329b,0x4624,0x57ad,0x6536,0x74bf,
    0x8c48,0x9dc1,0xaf5a,0xbed3,0xca6c,0xdbe5,0xe97e,0xf8f7,
    0x1081,0x0108,0x3393,0x221a,0x56a5,0x472c,0x75b7,0x643e,
    0x9cc9,0x8d40,0xbfdb,0xae52,0xdaed,0xcb64,0xf9ff,0xe876,
    0x2102,0x308b,0x0210,0x1399,0x6726,0x76af,0x4434,0x55bd,
    0xad4a,0xbcc3,0x8e58,0x9fd1,0xeb6e,0xfae7,0xc87c,0xd9f5,
    0x3183,0x200a,0x1291,0x0318,0x77a7,0x662e,0x54b5,0x453c,
    0xbdc b,0xac42,0x9ed9,0x8f50,0xfbef,0xea66,0xd8fd,0xc974,
    0x4204,0x538d,0x6116,0x709f,0x0420,0x15a9,0x2732,0x36bb,
    0xce4c,0xdfc5,0xed5e,0xfcd7,0x8868,0x99e1,0xab7a,0xbaf3,
    0x5285,0x430c,0x7197,0x601e,0x14a1,0x0528,0x37b3,0x263a,
    0xdec d,0xcf44,0xfddf,0xec56,0x98e9,0x8960,0xbbfb,0xaa72,
    0x6306,0x728f,0x4014,0x519d,0x2522,0x34ab,0x0630,0x17b9,
    0xef4e,0xfec7,0xcc5c,0xdd5,0xa96a,0xb8e3,0x8a78,0x9bf1,
    0x7387,0x620e,0x5095,0x411c,0x35a3,0x242a,0x16b1,0x0738,
    0xffcf,0xee46,0xdcdd,0xcd54,0xb9eb,0xa862,0x9af9,0x8b70,
    0x8408,0x9581,0xa71a,0xb693,0xc22c,0xd3a5,0xe13e,0xf0b7,
    0x0840,0x19c9,0x2b52,0x3adb,0x4e64,0x5fed,0x6d76,0x7cff,
    0x9489,0x8500,0xb79b,0xa612,0xd2ad,0xc324,0xf1bf,0xe036,
    0x18c1,0x0948,0x3bd3,0x2a5a,0x5ee5,0x4f6c,0x7df7,0x6c7e,
    0xa50a,0xb483,0x8618,0x9791,0xe32e,0xf2a7,0xc03c,0xd1b5,
    0x2942,0x38cb,0x0a50,0x1bd9,0x6f66,0x7eef,0x4c74,0x5dfd,
    0xb58b,0xa402,0x9699,0x8710,0xf3af,0xe226,0xd0bd,0xc134,
    0x39c3,0x284a,0x1ad1,0x0b58,0x7fe7,0x6e6e,0x5cf5,0x4d7c,
    0xc60c,0xd785,0xe51e,0xf497,0x8028,0x91a1,0xa33a,0xb2b3,
    0x4a44,0x5bcd,0x6956,0x78df,0x0c60,0x1de9,0x2f72,0x3efb,
    0xd68d,0xc704,0xf59f,0xe416,0x90a9,0x8120,0xb3bb,0xa232,
    0x5ac5,0x4b4c,0x79d7,0x685e,0x1ce1,0x0d68,0x3ff3,0x2e7a,
    0xe70e,0xf687,0xc41c,0xd595,0xa12a,0xb0a3,0x8238,0x93b1,
    0x6b46,0x7acf,0x4854,0x59dd,0x2d62,0x3ceb,0x0e70,0x1ff9,
    0xf78f,0xe606,0xd49d,0xc514,0xb1ab,0xa022,0x92b9,0x8330,
    0x7bc7,0x6a4e,0x58d5,0x495c,0x3de3,0x2c6a,0x1ef1,0x0f78
};

#define PPPINITFCS 0x0ffff /* initial "seed" for CRC */

```

SIZE A	CAGE CODE 88846	DWG NO 401002-SPEC	REV -
SCALE		SHEET 10	