Application Note AN-009

The BMD and BMDX File Formats

abaco SYSTEMS

Copyrights

Abaco Systems, Inc.
26 Castilian Drive, Suite B
Goleta, CA 93117
Main +1 805-965-8000 or +1 805-883-6101
Support +1 805-965-8000 or +1 805-883-6097

support@abaco.com (email)

https://www.abaco.com/products/avionics

Additional Resources

For more information, please visit the Abaco Systems website at:

www.abaco.com

Contents

Introduction

The BusTools/1553 analyzer software and the BusTools/1553-API store Bus Monitor data in binary files with a ".bmd" or ".bmdx" file extension.

These are called "BMD files" (Bus Monitor Data files) or "BMDX files" (Bus Monitor Data Files Extended). The purpose of this document is to explain these file formats to allow users to write their own programs to read and process these files.

This Application Note assumes a basic knowledge of 1553. For information on the MIL-STD-1553 protocol refer to the "MIL-STD-1553 Tutorial" document available from Abaco Systems. Additional information can also be found in MIL-HDBK-1553A.

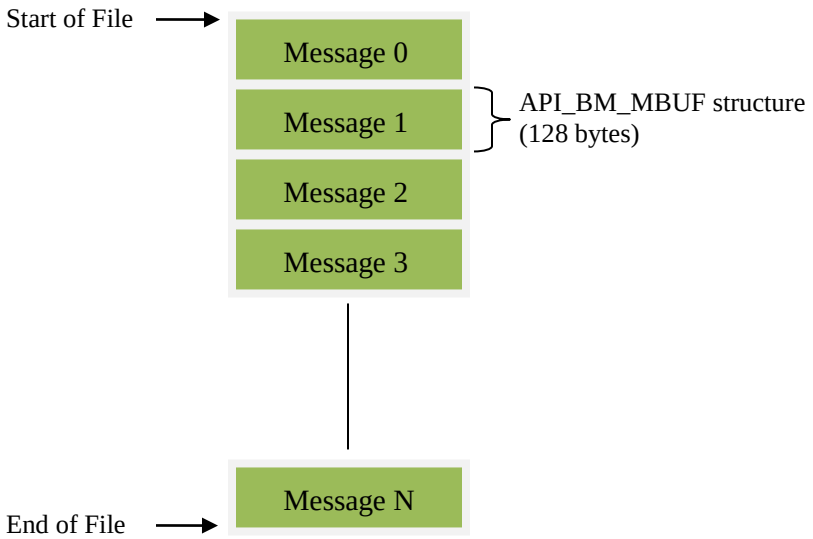
Note: BMDX is a newer file format and it replaces the older BMD format in the latest versions of the software. BusTools/1553 Bus Analyzer v7.30 and later use the BMDX file format but can still open and read log files in the BMD format to ensure backward compatibility.

BMD Files

The BMD file format is very simple. A BMD file is NOTHING but a series of records. There is no header or footer information. Each record in the file represents a 1553 message which corresponds to the 128-byte API_BM_MBUF structure as it is defined in the BusTools/1553-API version 6.44 or earlier.

Note: BMD is the format used by BusTools/1553 Bus Analyzer v7.20 and earlier.

The BMD File Structure



The API_BM_MBUF Structure (128-Bytes)

The BusTools/1553-API defines the API_BM_MBUF structure in the file “busapi.h”. This structure is also described in the “BusTools/1553-API Software Reference Manual” in Chapter 7 (Data Structures).

The definition is shown below:

```
#define BT1553_MBUFCOUNT 32 /* Data words in api_bm_mbuf struct */

typedef struct api_bm_mbuf
  {
  BT_U32BIT         messno;     // Message number (generated by API, 1-based)
  BT_U32BIT         int_status; // Interrupt status from board
  BT1553_TIME       time;       // Time of message (48-bits, 1 us LSB)
  BT1553_COMMAND    command1;   // 1553 command word #1 (Receive for RT-RT)
  BT_U16BIT         status_c1;  // 1553 command word #1 error status
  BT1553_COMMAND    command2;   // 1553 command word #2 (Transmit for RT-RT)
  BT_U16BIT         status_c2;  // 1553 command word #2 error status

  BT1553_BMRESPONSE response1;  // 1553 response time #1 (byte)
  BT1553_BMRESPONSE response2;  // 1553 response time #2 (byte)
  BT1553_STATUS     status1;    // 1553 status word #1 (Transmit for RT-RT or Broadcast RT-RT)
  BT_U16BIT         status_s1;  // 1553 status word #1 error status

  BT1553_STATUS     status2;    // 1553 status word #2 (Receive for RT-RT, NULL for BCST RT-RT)
  BT_U16BIT         status_s2;  // 1553 status word #2 error status

  BT_U16BIT         value[BT1553_MBUFCOUNT];   // 1553 data words
  BT_U8BIT          status[BT1553_MBUFCOUNT];  // 1553 status for data words
 }
API_BM_MBUF;
```

This structure provides all the information needed for a 1553 message, including time stamp, command words, status words, data words, response times, and error information. The 32-bit “int_status” field provides error/status flags for the message. The bits in this field are defined in the “BusTools/1553-API Software Reference Manual” in Chapter 7 (Data Structures), in the section for “Interrupt Enable / Message Status Bits (32 bit)”. Any errors on any word will be reflected in the 32-bit “int_status” word for the message. Errors are also reported for each word to identify exactly where errors occurred. Each command word and status word has an associated 16-bit error/status word that will reflect any errors on that specific word. Each data word has an associated 8-bit error/status word that will reflect any errors on that specific data word. The bits in these fields are defined in the “BusTools/1553-API Software Reference Manual” in Chapter 7 (Data Structures), in the section for “BM Word Status Bits (8/16 bit)”. The response time values (response1 and response2) are in units of 0.5 microseconds.

Example Program

Here is a very simple example program that reads a BMD file. For each message in the file, the program displays the message number and the command word.

```
#include <stdio.h>
#include "busapi.h"       // Header file from BusTools/1553-API v6.44 or earlier version


void main()
{
    FILE           *BMDfile;
    char           BMDfile_name[80];
    API_BM_MBUF    msg;
    unsigned long  num_msgs, i;

    printf("Input BMD filename: "); scanf("%s",BMDfile_name);

    BMDfile = fopen(BMDfile_name, "rb");
    if (BMDfile == NULL) {
        printf("ERROR OPENING BMD FILE %s\n",BMDfile_name);
    }
    else {

        fseek(BMDfile, 0, SEEK_SET);    // Make sure we are at beginning of file.
        fseek(BMDfile, 0, SEEK_END);    // Determine file size in records.
        num_msgs = ftell(BMDfile) / sizeof(API_BM_MBUF);
        printf("There are %d message records in the file.\n", num_msgs);
        fseek(BMDfile, 0, SEEK_SET); // Reset to beginning of file.

        // Read a message record.
        for (i=0; i<num_msgs; i++) {
            if ( fread( &msg, 1, sizeof(msg), BMDfile) != sizeof(msg) ) {
                printf("ERROR READING RECORD!\n");
                i = num_msgs;
            }
            else {

                // Display msg # and command word.
                printf("Message #: %d ", msg.messno);
               if (msg.command1.tran_rec)
                  printf("CMD1 = RT%02d TRANSMIT SA%02d WC%02d\n",
                         msg.command1.rtaddr,
                         msg.command1.subaddr,
                         msg.command1.wcount);
               else
                   printf("CMD1 = RT%02d RECEIVE SA%02d WC%02d\n",
                          msg.command1.rtaddr,
                          msg.command1.subaddr,
                          msg.command1.wcount);
            }
        }

        printf("Finished, closing file.\n");
        fclose(BMDfile);
     }
 }
```

BMDX Files

The BMDX file format was created to accommodate the new API_BM_MBUF structure as it was defined in the BusTools/1553-API v8.0 and later.

This new structure is very similar to previous one used in the BMD file format but with the two following improvements:

- 64-bit time tags (instead of 48-bit in the old structure). These time tags can either have a microsecond resolution (if the traffic was recorded with a board using firmware revision prior to v6.0) or a nanosecond resolution (if the traffic was recorded with boards using firmware revision v6.0 or later).

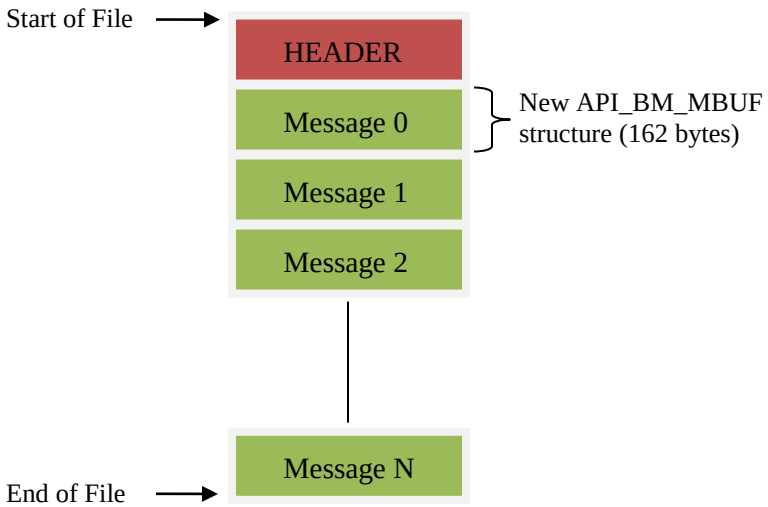
- 16-bit extended status/quality word (instead of 8-bit in the old structure) for each data word in the message.

In order to integrate these two changes, the BMD file format had to be modified and replaced by the new BMDX format.

The BMDX format is still very simple. It consists of a header (providing some information about the file) followed by a series of records. Each record after the header represents a 1553 message and corresponds to the new 162-byte API_BM_MBUF structure as it is defined in the BusTools/1553-API main header file for API versions 8.0 and later.

Note: BMDX is the format used by BusTools/1553 Bus Analyzer v7.30 and later.

The BMDX File Structure



The BMDX File Header

The BMDX header is 28-byte structure defined as follows:

```
// This is the header for the BMDX file format

typedef struct bmdx_header
{
        char       cTypeName[16]; //"BMDX"
        BT_U32BIT  uVersion;      // File version
        BT_U32BIT  uFormatInfo;   // Info about the content of the file
        BT_U32BIT  uReserved;     // Reserved Bits
}

BMDX_HEADER;
```

- The `cTypeName` parameter contains the format name and should always be “BMDX”.
- The `uVersion` parameter contains the value for the BMDX version.
- The `uFormatInfo` parameter provides some information about the records in the file:
    - Bit 0 - Indicates if the 64-bit time tags in the API_BM_MBUF records have a nanosecond or microsecond resolution (0 = microsecond, 1= nanosecond).
    - Bits 1 to 31 - Currently unused.
- The `uReserved` parameter is currently unused.

The New API_BM_MBUF Structure (162-Bytes)

The BusTools/1553-API (v8.0 and later) defines the API_BM_MBUF structure in the file "busapi.h". This structure is also described in the "BusTools/1553-API Software Reference Manual" in Chapter 7 (Data Structures). The definition is shown below:

```
#define BT1553_MBUFCOUNT 32 /* Data words in api_bm_mbuf struct */

typedef struct api_bm_mbuf
  {
  BT_U32BIT         messno;     // Message number (generated by API, 1-based)
  BT_U32BIT         int_status; // Interrupt status from board
  BT1553_TIME       time;       // Time of message (64-bits, 1 us  or 1ns LSB)
  BT1553_COMMAND    command1;   // 1553 command word #1 (Receive for RT-RT)
  BT_U16BIT         status_c1;  // 1553 command word #1 error status
  BT1553_COMMAND    command2;   // 1553 command word #2 (Transmit for RT-RT)
  BT_U16BIT         status_c2;  // 1553 command word #2 error status

  BT1553_BMRESPONSE response1;  // 1553 response time #1 (byte)
  BT1553_BMRESPONSE response2;  // 1553 response time #2 (byte)
  BT1553_STATUS     status1;    // 1553 status word #1 (Transmit for RT-RT or Broadcast RT-RT)
  BT_U16BIT         status_s1;  // 1553 status word #1 error status

  BT1553_STATUS     status2;    // 1553 status word #2 (Receive for RT-RT, NULL for BCST RT-RT)
  BT_U16BIT         status_s2;  // 1553 status word #2 error status

  BT_U16BIT         value[BT1553_MBUFCOUNT];   // 1553 data words
  BT_U16BIT         status[BT1553_MBUFCOUNT];  // 1553 status for data words (16-bit words)
 }
API_BM_MBUF;
```

As was the case for the BMD format, this structure provides all the information needed for a 1553 message, including time stamp (64-bit), command words, status words, data words, response times, and error information. The 32-bit "int_status" field provides error/status flags for the message. The bits in this field are defined in the "BusTools/1553-API Software Reference Manual" in Chapter 7 (Data Structures), in the section for "Interrupt Enable / Message Status Bits (32 bit)". Any errors on any word will be reflected in the 32-bit "int_status" word for the message. Errors are also reported for each word to identify exactly where errors occurred. Each command word, data word, and status word has an associated 16-bit error/status word that will reflect any errors on that specific word. The bits in these fields are defined in the "BusTools/1553-API Software Reference Manual" in Chapter 7 (Data Structures), in the section for "BM Word Status Bits (8/16 bit)". The response time values (response1 and response2) are in units of 0.5 microseconds.

Example Program

Here is a simple example program that reads a BMDX file. For each message in the file, the program displays the message number and the command word.

```
#include <stdio.h>
#include "busapi.h"       // Header file from BusTools/1553-API v8.00 or later

typedef struct bmdx_header{
     char       cTypeName[16];
     BT_U32BIT  uVersion;
     BT_U32BIT  uFormatInfo;
     BT_U32BIT  uReserved;
}
BMDX_HEADER;

#define BMDX_HEADER_SIZE sizeof(BMDX_HEADER)

void main()
{
    FILE           *BMDXfile;
    char           BMDXfile_name[80];
    API_BM_MBUF    msg;
    unsigned long  num_msgs, i;

    printf("Input BMDX filename: "); scanf("%s",BMDXfile_name);

    BMDXfile = fopen(BMDXfile_name, "rb");
    if (BMDXfile == NULL) {
        printf("ERROR OPENING BMD FILE %s\n",BMDXfile_name);
    }
    else {

        fseek(BMDXfile, 0, SEEK_SET);    // Make sure we are at beginning of file.
        fseek(BMDXfile, 0, SEEK_END);    // Determine file size in number of records.
        num_msgs = (ftell(BMDXfile) - BMDX_HEADER_SIZE) / sizeof(API_BM_MBUF);
        printf("There are %d message records in the file.\n", num_msgs);

        fseek(BMDXfile, BMDX_HEADER_SIZE, SEEK_SET); // Reset to beginning of the BM records.

        // Read a message record.
        for (i=0; i<num_msgs; i++) {
            if ( fread( &msg, 1, sizeof(msg), BMDXfile) != sizeof(msg) ) {
                printf("ERROR READING RECORD!\n");
                i = num_msgs;
            }
            else {

                // Display msg # and command word.
                printf("Message #: %d ", msg.messno);
               if (msg.command1.tran_rec)
                  printf("CMD1 = RT%02d TRANSMIT SA%02d WC%02d\n",
                         msg.command1.rtaddr,
                         msg.command1.subaddr,
                         msg.command1.wcount);
               else
                   printf("CMD1 = RT%02d RECEIVE SA%02d WC%02d\n",
                          msg.command1.rtaddr,
                          msg.command1.subaddr,
                          msg.command1.wcount);
            }
        }

        printf("Finished, closing file.\n");
        fclose(BMDXfile);
    }
 }
```