DSTO

Functions for Writing and Reading Time History Data

Geoff Brian

DSTO-GD-0183

APPROVED FOR PUBLIC RELEASE

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Functions for Writing and Reading Time History Data

Geoff Brian

Air Operations Division
Aeronautical and Maritime Research Laboratory

DSTO-GD-0183

ABSTRACT

The manipulation of time history data is one of the most common activities conducted when analysing aircraft flight behaviour and performance, and as a result there exists a multitude of data formats individualised for specific applications. Air Operations Division (AOD), of the Defence Science and Technology Organisation, have chosen to support two time history data formats defined by NASA Dryden Flight Research Center for development of flight behaviour and performance applications. A suite of functions for writing and reading the selected NASA time history data formats have been developed at AOD. These functions may be incorporated into analysis applications to reduce development time for new software, and to improve the sharing of data between applications.

RELEASE LIMITATION

Approved for public release

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

DSTO Aeronautical and Maritime Research Laboratory
PO Box 4331
Melbourne Victoria 3001

Telephone: (03) 9626 8111
Fax: (03) 9626 8999

AR No. AR-010-55[illegible]4
June 1998

APPROVED FOR PUBLIC RELEASE

Functions for Writing and Reading Time History Data

Executive Summary

Aircraft behaviour analysis and performance estimation applications utilise time history data for control parameters, such as pilot stick movements, and the recording of aircraft responses and trajectory data. The manipulation of time history data has become one of the most common activities conducted when analysing aircraft behaviour, and as a result there exist a multitude of distinct data formats designed for specific applications.

With the establishment of aircraft flight behaviour research and aircraft performance estimation in Air Operations Division (AOD), experience was gained with applications for the analysis of flight trial data developed by NASA. Together with these applications, NASA defined a number of formats for the storage of time history data which were efficient at minimising file sizes and included features permitting fast access to data. AOD chose to support two of the time history data formats defined by NASA when developing flight behaviour and performance estimation applications. The two formats were an ASCII character format which presented data in a readable form, and a compressed binary format designed to minimise file size and maximise data access speed.

A suite of functions for writing and reading the selected NASA time history data formats have been developed in AOD. These functions permit applications to manipulate data files of either format, and access data with reference to a signal name. Consequently, prior knowledge of the format and order of data storage in a file is not required. In addition, the functions permit compressed binary data files to be independent of computer type and operating system. The suite of functions for writing and reading time history data has reduced development time for new aircraft behaviour and performance applications in AOD, and improved the sharing of data between applications.

Authors

Geoffrey J. Brian
Air Operations Division

Geoff Brian commenced employment at the Aeronautical and Maritime Research Laboratory in 1989, after graduating from the University of New South Wales with a degree in Aeronautical Engineering, Honours Class 1. During this time he has been involved with the analysis of flight data for the development of a comprehensive flight dynamic model of the F-111C aircraft; the creation of applications for the estimation of aircraft performance; and involvement with a performance flight trial of a F-111C aircraft fitted with Pratt & Whitney TF30-P-109 engines. Mr. Brian was attached to the Royal Australian Air Force - Aircraft Research and Development Unit in 1992, where he was involved with a performance flight trial of an air-to-air refuelling capable Boeing 707 aircraft. Mr. Brian is currently the manager of aircraft performance estimation activities at AMRL.

Contents

Appendices D & E have been reproduced on Microfiche

Nomenclature

AOD	Air Operation Division
ASCII	American Standard Code for Information Interchange
DSTO	Defence Science and Technology Organisation
NASA	National Aeronautics and Space Administration
$Time_{stamp}$	Encoded time stamp

1. Introduction

Applications which analyse aircraft behaviour often utilise time history data for control parameters, such as pilot stick movements, and the recording of aircraft responses and trajectory data. A multitude of distinct data formats have been defined for use with different applications, restricting easy sharing of data, and leading to a proliferation of data format conversion applications.

Through collaborative activities with NASA, AOD have gained experience using applications for the analysis of flight trial data and aircraft behaviour. AOD have extensively used NASA applications for the manipulation of time history data [1]; aircraft aerodynamic parameter estimation [2]; and plotting [3]. Together with these applications, NASA have defined a number of formats for the storage of time history data. AOD chose to support two of the NASA time history data formats when developing flight behaviour and performance estimation applications. The two formats were an ASCII character format presenting data in a readable form, and a compressed binary format which was designed to minimise the file size and maximise data access speed. A detailed description of the two formats is presented.

A suite of functions for writing and reading the selected NASA time history data formats have been developed at AOD. These functions permit applications to manipulate data files of either format without prior knowledge of the format type. Knowledge of the order of data storage in a file is not required as data may be accessed with reference to a signal name. Also, the functions permit compressed binary data files to be independent of computer type and operating system. Each of the functions for writing and reading time history data are described, together with examples of how the functions are invoked from within an application.

2. Data File Formats

AOD have chosen to support two of the time history formats defined by NASA when developing applications for analysing aircraft behaviour and performance estimation. The ASCII format was chosen since information is presented in a readable form, as shown in figure 1. This format contains a *header* section detailing the number of channels of data stored in a file and the signal name for each channel. Time history data for each channel, and the corresponding time are stored in the *data* section. A detailed description of the ASCII format is presented in Appendix A.

```
Header Section
format  asc 1   .1
nChans        24
names           axagm           ayagm           azagm           pm
qm              rm              altftm          xmachm          vtasm
alpham          betam           pdotm           qdotm           rdotm
thetam          phim            delhm           delam           delrm
delspm          stklnm          stkltm          rpm             delctm
data001

Data Section
     0.000          0.14350000000000    -.40000000000000E-01 1.0876000000000
-.19700000000000E-01-.20000000000000E-03-.10000000000000E-02 5489.0560000000
0.78980000000000     864.48690000000     2.8617000000000     0.24040000000000
-.10890000000000    -.94300000000000E-010.13030000000000     0.96010000000000
-1.7540000000000    0.37470000000000    0.13920000000000    -.11870000000000
-.45250000000000    -.42620000000000    0.00000000000000E+000.26340000000000
0.00000000000000E+00
     0.017          0.17140000000000    -.21300000000000E-01 1.0742000000000
-.16900000000000E-01-.20000000000000E-03-.10000000000000E-02 5486.6080000000
0.78970000000000     864.45960000000     2.9411000000000     0.24040000000000
-.17800000000000E-01-.51200000000000E-01-.35800000000000E-010.78400000000000
-1.8417000000000    0.37470000000000    0.13920000000000    -.62000000000000E-02
-.43080000000000    -.41720000000000    0.00000000000000E+000.26340000000000
0.00000000000000E+00
     0.033          0.14960000000000    -.13100000000000E-01 1.0528000000000
-.14200000000000E-01-.20000000000000E-03-.10000000000000E-02 5486.6080000000
0.78970000000000     864.45960000000     2.9411000000000     0.24040000000000
0.12870000000000    0.11150000000000    -.43300000000000E-010.78400000000000
-1.8417000000000    0.38580000000000    0.12810000000000    0.38900000000000E-01
-.43080000000000    -.41720000000000    0.53000000000000E-020.26340000000000
0.00000000000000E+00
```

Figure 1: Sample ASCII Time History File.

The compressed binary format was chosen because it minimises file size and supports fast random access to the data. Information detailing the file format, the number of channels of data stored in the file, signal names for each channel and comments pertaining to the data stored in the file, are contained in a *header* section as shown in Figure 2. An *index* section contains information which is used to increase the access speed of the file when reading data. Time history data and the corresponding time, encoded as a time stamp ($Time_{stamp}$), are stored in the file following the header and index sections. A detailed description of the compressed binary format is presented in Appendix B.

Applications using the functions developed at AOD are capable of manipulating files of either the ASCII or compressed binary time history format without prior knowledge of the particular file format. Data may be accessed with reference to a signal name, and therefore no prior knowledge of the order of data storage is required. The functions also permit the compressed binary files to be independent of the machine type and operating system on which they were created. Consequently, data may be easily shared between applications without compromising the size of time history files.

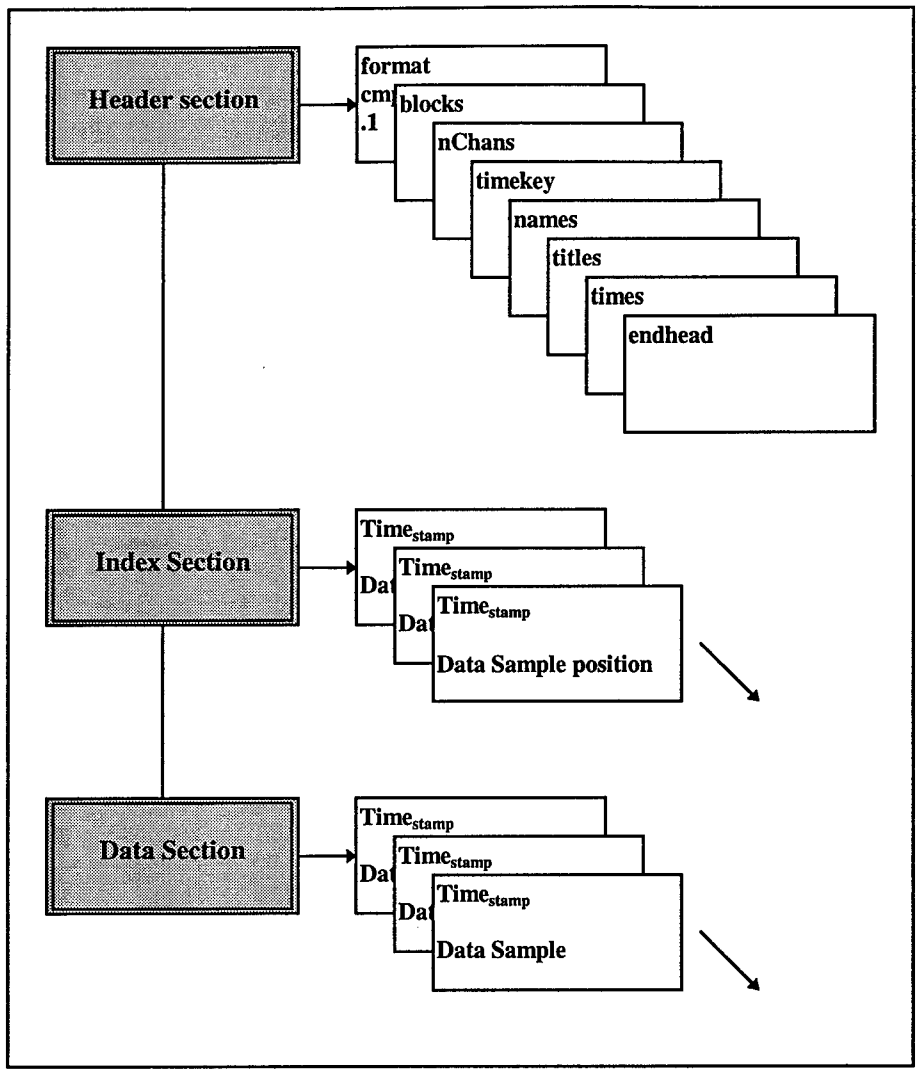

Figure 2 : Compressed Time History File Format

3. Functions for Writing

The set of functions for writing time history files using the formats detailed in Section 2 are described in this section. The set includes a function for opening a new file to receive time history data (OpenW), a function for writing data to the file (FWrite), and a function to close the file (CloseW). An example of how each function may be invoked is presented using C programming language conventions [5], although the functions may be accessed by applications written using other programming languages. A sample application in FORTRAN 77 is presented in Appendix C.

3.1 OpenW : Open a File for Writing.

This function is used to open and initialise files for writing time history data. It has the following prototype† definition:

int OpenW(int *, char *, int *, char *, char *);

and is invoked using the statement:

OpenW(&‡Unit_no, name, &nChans, Sigs, format)

The parameters enclosed in brackets are defined as:

- Unit_no : An identifier affiliated with the file specified by the ***name*** parameter.
- name : The name of the file being opened.
- nChans : The number of channels of data to be written to the file.
- Sigs : A list of signal names for each channel of data.
- format : The format style to be used when writing data to the file.

The format variable has an entry of either ***asc1*** if an ASCII file is to be written, or ***cmp3*** for a compressed binary file.

A header section is written to the newly opened file detailing the number of channels of data in the file, and the signal names for each data channel. A value of "1" is returned by the function if it successfully opens a file, otherwise "0" is returned. This function will fail if the format for the file is invalid, the file identifier is being used for another write file, or the file is already open.

† The prototype definition is a C programming convention where the number and type of arguments passed through to a function (enclosed in brackets) are defined. The type of the value returned by the function is also defined.

‡ A '&' symbol preceding a parameter name, such as Unit_no, implies that the memory address of the data is be passed to the function, as opposed to the value of the data. Within the function, the parameter will be referenced as a pointer to the data.

3.1.1 Example of the OpenW function

```
int    Unit_no, nChans;
char   *Sigs;
Unit_no  = 1;
nChans   = 20;
Sigs :: points to a list of signal names.

if ((OpenW(&Unit_no, "test.txt", &nChans, Sigs, "asc1")) == FALSE)
  printf("Error Opening File for writing");
else
  {....}
```

3.2 FWrite : Write Data to a File.

This function is used to write data, for a discrete time, to a file which has previously been opened using the **OpenW** function. It has the following prototype definition:

void FWrite(int *, double *, double *);

and is invoked using the statement:

FWrite(&Unit_no, &time, &data);

The parameters enclosed in brackets are defined as:

- Unit_no : The file identifier specified in the call to **OpenW**.
- time : The time corresponding to the data being written.
- data : An array containing the channel data.

Data for each channel are written to the file, denoted by **Unit_no,** using the format specified during the call to the **OpenW** function.

3.2.1 Example of the FWrite function

```
int    Unit_no;
int    nChans;
char   *Sigs;
double time;
double Data_R[100];

Unit_no  = 1;
nChans   = 20;
Sigs :: points to a list of signal names.

if ((OpenW(&Unit_no, "test.txt", &nChans, Sigs, "asc1")) == FALSE)
  printf("Error Opening File for writing");
else
  {
  <loop to write data >
    {
    ....
    time = ?????;
    Data_R :: points to a list of channel data.
    FWrite(&Unit_no, &time, &Data_R);
  }
  ....
}
```

3.3 CloseW : Close a Write File

This function is used to close a time history data file that has previously been opened for writing. It has the following prototype definition:

void CloseW(int *);

and is invoked using the statement:

CloseW(&Unit_no);

The parameter enclosed in brackets is the file identifier specified in the call to **OpenW.**

3.3.1 Example of the CloseW function

```
int    Unit_no;
int    nChans;
char   *Sigs;

Unit_no  = 1;
nChans   = 20;
Sigs :: points to a list of signal names.

if ((OpenW(&Unit_no, "test.txt", &nChans, Sigs, "asc1")) == FALSE)
  printf("Error Opening File for writing");
else
  {
  ....
  ....
  CloseW(&Unit_no);
}
```

4. Functions for Reading

The set of functions for reading time history files, corresponding to the formats detailed in Section 2, are described in this section. The set includes a function to open an existing time history file (OpenR), a function to retrieve the list of signal names for data in the file (RSigs), functions to specify what data is to be read from the file (SigsR & ChansR), functions to sequentially or randomly retrieve data from the file (FRead & FSeek), a function to rewind the file (RewR), and a function to close the file (CloseR). An example of how each function may be invoked is presented using C programming language conventions, although the functions may be accessed by applications written using other programming languages.

4.1 OpenR : Open a File for Reading.

This function is used to open and initialise files from which time history data will be read. It has the following prototype definition:

int OpenR(int *, char *, int *);

and is invoked using the statement:

OpenR(&Unit_no, name, &nChans);

The parameters enclosed in brackets are defined as:

- Unit_no : An identifier affiliated with the file specified by the *name* parameter.
- name : The name of the file being opened.
- nChans : The number of data channels available in the file.

This function will read the header section of the file which has been opened. The number of data channels available in the file is returned through the variable **nChans**. A list of the signal names for each data channel is stored in memory and can be accessed using the function **RSigs**. A value of "1" is returned by the function if it successfully opens a file, otherwise "0" is returned. This function will fail if the file does not exist, the file is already open, the file data format is invalid, or the file identifier is being used by another read file.

4.1.1 Example of the OpenR function

```
int    Unit_no;
int    nChans;

Unit_no  = 1;

if ((OpenR(&Unit_no, "test.txt", &nChans)) == FALSE)
  printf("Error Opening File for reading");
else
  {....}
```

4.2 RSigs : Return Signals Avaliable in the Read File.

This function is used to return a list of the signals names for data available in a time history file which has previously been opened using the **OpenR** function. The signal name list is compiled from data stored in memory when the file was opened. The **RSigs** function has the following prototype definition:

void RSigs(int *Unit_no, char *Sigs);

and is invoked using the statement:

RSigs(&Unit_no, Sigs);

The parameters enclosed in brackets are defined as:

- Unit_no : The file identifier specified in the call to **OpenR**.
- Sigs : A pointer to an array which accepts the signal name list.

4.2.1 Example of the RSigs function

```
int    j;
int    Unit_no;
int    nChans;
char   *Chan_R, *TmpSigs;

Unit_no  = 1;

if ((OpenR(&Unit_no, "test.txt", &nChans)) == FALSE)
  printf("Error Opening File for reading");
else
  {

  Chan_R = (char *)malloc(nChans*20*sizeof(char));

  RSigs(&Unit_no, Chan_R);

  TmpSigs = Chan_R;
  for (j=0; j<nChans; ++j)
    {
    printf("Signals :: %s\n", TmpSigs);
    TmpSigs = next_word(TmpSigs);
  }
  ....

/* Free Memory allocated for signal list after finished with */
  free(Chan_R);
  ....
}
```

4.3 SigsR : Set Signals to be Read.

This function is used to set which channels of data will be read from the time history file by specifying the channel signal names. The selected signal names are cross-referenced with information stored in memory detailing the column number for each channel of data. A list of column numbers for data to be read is constructed and stored in memory for future use by the **FRead** or **FSeek** functions. The **SigsR** function has the following prototype function:

void SigsR(int *, char *, int *);

and is invoked using the statement:

SigsR(&Unit_no, Sigs, &nSigs);

The parameters enclosed in brackets are defined as:

- Unit_no : The file identifier specified in the call to **OpenR**.
- Sigs : A pointer to a signal name list of data to be read from the file.
- nSigs : The number of data channels to be read from the data file.

4.3.1 Example of the SigsR function

```
int    Unit_no;
int    nChans;
int    nSigs;
double time;
double Data_R[100];
char   *SigsRead = "zero\0three\0eight\0nineteen\0";

Unit_no  = 1;
nSigs    = 4;

if ((OpenR(&Unit_no, "test.txt", &nChans)) == FALSE)
  printf("Error Opening File for reading");
else
  {
/* Selecting the data to be read from the file by signal name */
  SigsR(&Unit_no, SigsRead, &nSigs);

  while ((FRead(&Unit_no, &time, &Data_R)) != FALSE)
  time :: correspond to the time of the data record.
  Data_R :: points to a list of channel data.
  ....
}
```

4.4 ChansR : Set Channels to be Read.

This function is used to set which channels of data will be read from the time history file by specifying the data column number. A list of column numbers for data to be read is constructed and stored in memory for future use by the **FRead** or **FSeek** functions. The **ChansR** function has the following prototype function:

void ChansR(int *, int *, int *);

and is invoked using the statement:

ChansR(&Unit_no, &chans, &nSigs);

The parameters enclosed in brackets are defined as:

- Unit_no : The file identifier specified in the call to **OpenR**.
- chans : A pointer to a column number list of data to be read from the file.
- nSigs : The number of data channels to be read from the data file.

4.4.1 Example of the ChansR function

```
int    Unit_no;
int    nChans;
int    nSigs;
int    Data_I[4] = {1,4,6,12};
double time;
double Data_R[100];

Unit_no  = 1;
nSigs    = 4;

if ((OpenR(&Unit_no, "test.txt", &nChans)) == FALSE)
  printf("Error Opening File for reading");
else
  {
/* Selecting the data to be read from the file by column number */
  ChansR(&Unit_no, &Data_I, &nSigs);

  while ((FRead(&Unit_no, &time, &Data_R)) != FALSE)
  time :: correspond to the time of the data record.
  Data_R :: points to a list of channel data.
  ....
}
```

4.5 FRead : Read Data from a File.

This function is used to read data, for a discrete time, from a file which has previously been opened using the **OpenR** function. The first time **FRead** is invoked, data associated with the first time entry will be read from the file. Further calls will read data associated with successive time entries. **FRead** may be invoked without previously calling **SigsR** or **ChansR** to select which channels of data to read, since the **OpenR** function creates a default column number list consisting of all the available channels of data. The **FRead** function has the following prototype definition:

int FRead(int *, double *, double *);

and is invoked using the statement:

FRead(&Unit_no, &time, &data);

The parameters enclosed in brackets are defined as:

- Unit_no : The file identifier specified in the call to **OpenR**.
- time : The time corresponding to the data which was read from the file.
- data : An array which accepts the channel data.

Channel data read from the file are returned through the variable **data**, and the corresponding time returned through the variable **time**. A value of "1" is returned by the function if it successfully reads data from the file, otherwise "0" is returned. The function will fail if the end of the file has been reached.

4.5.1 Example of the FRead function

```
int    Unit_no;
int    nChans;
double time;
double Data_R[100];

Unit_no  = 1;

if ((OpenR(&Unit_no, "test.txt", &nChans)) == FALSE)
  printf("Error Opening File for reading");
else
  {
  while ((FRead(&Unit_no, &time, &Data_R)) != FALSE)
  time :: correspond to the time of the data record.
  Data_R :: points to a list of channel data.
  ....
}
```

4.6 FSeek : Read Data Corresponding to a Nominated Time.

This function reads data corresponding to a nominated time. It has the following prototype definition:

int FSeek(int *, double *, double *, double *);

and is invoked using the statement:

FSeek(&Unit_no, &tSeek, &time, &data);

The parameters enclosed in brackets are defined as:

- Unit_no : The file identifier specified in the call to **OpenR**.
- tSeek : The nominated time at which data to is to be read from the file.
- time : The time corresponding to the data which was read from the file.
- data : An array which accepts the channel data.

During execution of this function the file position indicator is repositioned such that the information returned through the variable **data** corresponds to a time equal to or greater than the nominated time **tSeek**. The time corresponding to data that was read from the file is returned through the variable **time**. A value of "1" is returned by the function if it successful reads data from the file, otherwise "0" is returned. The function will fail if the time **tSeek** does not occur between the start and end times for data stored in the file.

4.6.1 Example of the FSeek function

```
int    Unit_no;
int    nChans;
double time, tSeek;
double Data_R[100];

Unit_no  = 1;
tSeek    = 20.0;

if ((OpenR(&Unit_no, "test.txt", &nChans)) == FALSE)
  printf("Error Opening File for reading");
else
  {
  if ((FSeek(&Unit_no, &tSeek, &time, &Data_R)) != FALSE)
    {
    while ((FRead(&Unit_no, &time, &Data_R)) != FALSE)
    time :: correspond to the time of the data record.
    Data_R :: points to a list of channel data.
    ....
  }
}
```

4.7 RewR : Rewind a Read File.

This function is used to reset the file such that the next data record read from the file will be that beginning the data section. It has the following prototype definition:

void RewR(int *);

and is invoked using the statement:

RewR(&Unit_no);

The parameter enclosed in brackets is the file identifier specified in the call to **OpenR.**

4.7.1 Example of the RewR function

```
int    Unit_no;
int    nChans;
double time;
double Data_R[100];

Unit_no  = 1;

if ((OpenR(&Unit_no, "test.txt", &nChans)) == FALSE)
  printf("Error Opening File for reading");
else
  {
  while ((FRead(&Unit_no, &time, &Data_R)) != FALSE)
    {
    time :: correspond to the time of the data record.
    Data_R :: points to a list of channel data.
    ....
    ....
  }

/* Rewind to start of Data Section */
    RewR(&Unit_no);
    if ((FRead(&Unit_no, &time, &Data_R)) != FALSE)
      {
      ....
    }
  }
```

4.8 CloseR : Close a Read File.

This function is used to close a time history data file that has previously been opened for reading. It has the following prototype definition:

void CloseR(int *);

and is invoked using the statement:

CloseR(&Unit_no);

The parameter enclosed in brackets is the file identifier specified in the call to **OpenR.**

4.8.1 Example of the CloseR function

```
int    Unit_no;
int    nChans;

Unit_no  = 1;

if ((OpenR(&Unit_no, "test.txt", &nChans)) == FALSE)
  printf("Error Opening File for reading");
else
  {
  ....
  ....
  CloseR(&Unit_no);
}
```

5. Incorporating Time History Functions in an Application

To permit access to time history data using the functions described in sections 3 and 4, files containing source code for the functions must be bound into the application. Table 1 lists the files containing source code for the time history functions, together with a description of the contents of each file. The time history functions have been developed using the C programming language, and listings of the files are presented in Appendix D. A list of associated include files for the source code is also shown in Table 1, and a listing of each include file is presented in Appendix E.

Table 1: Source Code Files for Time History Functions

Source Code Files	
File_1.c	Contains top level interface functions for writing and reading time history data files
File_2.c	Contains format dependent functions for writing and reading time history data files.
File_3.c	Contains functions for manipulation of stored data relating to each time history data file.
File_4.c	Miscellaneous functions.
File_5.c	Contains functions to access binary data format files.
Included Header Files	
common.h errstrn.h fmttype.h extrout.h file_2.h file_3.h file_4.h file_5.h macro.h macroerr.h os_comp.h	

Information stored in the files **os_comp.h** and **extrout.h,** is used by the time history functions to interface correctly with an application's source code. The **os_comp.h** file contains identifiers denoting the language chosen for the development of the application and the operating system under which the application is being developed. The **extrout.h** file contains prototype definitions of the time history functions discussed in Sections 3 and 4.

For an application written using the C programming language to interface with the time history functions, the two files are required to be included into the application's source code. This may be achieved by inserting the following lines:

#include <os_comp.h>
#include <extrout.h>

Other programming languages may use different techniques for interfacing with external functions. Information on these techniques may be found in references on the programming languages being used for application development.

There are many ways in which the files containing source code for the time history functions may be bound into an application. These depend on the operating system under which the application is being developed, the programming language which is being used to developed the application, and any application development software tools that may be used, e.g. Microsoft™ Developer Studio. One possible method for incorporating the time history source code files is described with the assistance of the simple Makefile presented in Figure 3.

```
# Simple Makefile to compile application testapp.c
# which incorporates time history functions

# Compiling Flags
.SUFFIXES: .c.o .c .o
CC    =       cc

# Directory Locations
# Time History Include Files
THINC =       ./common/include/timeh
# Time History Source Code Files
THSRC =       ./common/source/timeh

# Compile source code to create object files
${CC} -I${THINC} -c ${THSRC}/file_1.c
${CC} -I${THINC} -c ${THSRC}/file_2.c
${CC} -I${THINC} -c ${THSRC}/file_3.c
${CC} -I${THINC} -c ${THSRC}/file_4.c
${CC} -I${THINC} -c ${THSRC}/file_5.c

${CC} -I${THINC} -c ${THSRC}/testapp.c

# Bind object files to create executable file testapp.exe
${CC} testapp.o file_1.o file_2.o file_3.o file_4.o file_5.o -o testapp

# End of simple Makefile
```

Figure 3 : Example Makefile for including time history functions in an application

In this example the include files, listed in Table 1, are assumed to reside in the directory assigned to the **THINC** variable. Source code files for the time history functions are assumed to reside in the directory assigned to the **THSRC** variable. Upon execution of the Makefile, the time history source code files, and the application, are compiled to create *object* files using the C language compiler cc. The object files are then bound to create an executable file for the application.

Information on more efficient compilation techniques may be found in references on the programming languages being used for application development.

6. Conclusion

A suite of functions for writing and reading time history data have been developed by AOD. These functions support an ASCII character time history data format, as well as a compressed binary format. The ASCII format presents data in a readable form, while the compressed binary format has been designed to minimise file size and maximise the access speed to data. Both of these formats were defined by NASA for development of flight behaviour and aircraft performance applications.

The functions discussed in this report permit applications to manipulate time history data of either format, and access data with reference to a signal name. In addition, the functions permit time history data files to be independent of computer type and operating system. The suite of functions for writing and reading time history data may be incorporated into analysis applications to reduce development time for new software, and improve the sharing of data between applications.

7. References

[1] Maine R.E. (1987), Manual for GetData Version 3.1 - A FORTRAN Utility Program for Time History Data., NASA Technical Memorandum 88288, Edwards, California.

[2] Murray J.E., Maine R.E. (1987), pEst Version 2.1 User's Manual., NASA Technical Memorandum 88280, Edwards, California.

[3] Vernon T., (1992), Xplot - A Utility for Plotting X-Y Data User Manual / Command Reference., PRC Inc., NASA Ames Dryden Flight Research Center, Edwards, California.

[4] X3.4-1977 : American National Standard Code for Information Exchange

[5] Kelley A. and Pohl I. (1984), A Book on C - Programming in C (2nd Ed.), The Benjamin/Cummings Publishing Company, Inc., California. ISBN 0-8053-0060-0

[6] IEEE Std 754-1985 (R1990) IEEE Standard for Binary Floating-Point Arithmetic (ANSI) [1-55937-653-8] [SH10116-NYF]

Appendix A : ASCII Format

The ASCII format presents time history data in a readable form. Descriptive information and time history data stored in a file is organised into *records* each having a length of 80 characters. The file contains a header section and a data section. Figure 1 illustrates the format of a ASCII time history data file.

Header Section

The header section contains a series of records storing information relating to the format of the file, the number of channels of data stored in the file, and the signal name for each channel of data. Information stored in the header section is preceded by a tag identifying the data which follows. All character data are left justified within their fields, while numerical data in the header are right justified. A description of each record follows.

Format Record

The first header record identifies the format of data stored in the file, and is used to automate the management of different file formats. The tag for this record is **'format'**, which is followed by the format description **'asc 1'**. The identifier **'.1'** represents the version of the ASCII format definition used for storing data in a file; however, this information is not used when accessing time history files.

Columns	Field Format	Value	
1-8	Character	**format**	
9-16	Character	**asc␣1**	*{ The symbol "␣" represents a space }*
17-18	Character	**.1**	

Number of Channels Record

The number of channels of data that are available in a file is specified in the second header record. The tag for this record is **'nChans'**, which is followed by the number of channels.

Columns	Field Format	Value
1-8	Character	**nChans**
9-16	Integer	Number of channels

Signal Name Record(s)

The third header record initiates the list of signal names for the channels of data stored in the file. Signal names are limited to 16 characters in length, and the list continues across as many records as required to document all names. The format of the first signal name record differs from following records as it contains the tag **'names'** as well as the first 4 signals names. Following signal name records each contain 5 names per record, except for the last record which may contain less.

Columns	Field Format	Value	
1-8	Character	**names**	*{1st Record}*
9-16	Blank		
17-32	Character	Name of signal 1	
33-48	Character	Name of signal 2	
49-64	Character	Name of signal 3	
65-80	Character	Name of signal 4	
1-16	Character	Name of signal	*{Other Records}*
17-32	Character	Name of signal	
33-48	Character	Name of signal	
49-64	Character	Name of signal	
65-80	Character	Name of signal	

Data Header Record

This record is used to indicate the end of the header section. The tag used to define this record is '**data001**', and no data is associated with this record.

Data Section

The remainder of the file contains data for each channel. The data are stored as floating point numbers using a format field width of 20 characters, with a maximum of 14 decimal places. Data, for each discrete time, are stored in the same order as the signal names, using as many records as required to document all channels. The first record for each discrete time differs from the following records as it contains the time corresponding to the data sample in the first field of 20 characters, and is followed by 3 data entries. Following records each contain 4 data entries, except for the last record which may contain less.

Appendix B : Compressed Binary Format

The compressed file format has been designed to minimise file size and permit fast access to data in time history files. The following conventions are used for information stored in a compressed file.

- Character data are encoded using the ASCII 8 bit character set standard, [4].

- 8, 16, 24 and 32 bit integer data are encoded with the most significant byte stored first. This is followed by the lesser significant bytes, with the least significant byte last. Negative integers are encoded using Two's Complement integer notation which is defined in [5].

- Floating point data are encoded using the IEEE Standard 754-1985, [6], with the most significant byte stored first, followed by lesser significant bytes, and the least significant byte last. 24, 32 and 64 bit floating point data are stored in the compressed time history files, with the 24 bit floating point data representing 32 bit floating point data with the least significant byte removed.

The file contains a header section, an index section, and a data section. Each section is comprised of a series of *blocks*, with a block consisting of a number§ of consecutive bytes of information. The first byte in a block indicates whether information contained within that block relates to the header section, the index section or the data section of a file. The next three bytes contain the number of the next block completely filled with information for the section denoted by the first byte. A block with binary zero entries for these three bytes is the last block for that section type. Blocks which are not completely filled with information are padded using binary zero entries for each unused byte. Figure 2 illustrates the form of the compressed format time history data file.

Header Section

The header section contains information detailing the file format, the number of blocks of information stored in the file, the number of channels of data stored in the file, signal names for each channel of data, optional titles, and the start time and end time for data stored in the file. The length and the type of information stored for a particular entry in the header section are encoded in the first three bytes of that entry. All character data are left justified in their fields, while numerical data is stored using an equivalent binary format. A description of each header record follows.

§ The default size of a block is 2048 bytes. The block size, in bytes, is stored in the **blocks** entry of the **header** section.

Format Entry

The first entry identifies the format of data stored in the file, and is used to automate the management of different file formats. The tag for this entry is '**format**', and is followed by the format description '**cmp 3**'. The identifier '**.1**' represents the version of the compressed binary format definition used for storing data in a file; however, this information is not used when accessing time history files.

Bytes	Field Format	Value
1-2	Integer	Length of Entry (27)
3	Integer	Entry Type
4-11	Character	**format**
12-19	Character	**cmp␣3** *{ The symbol "␣" represents a space }*
20-27	Character	**.1**

Blocks

The second entry stores the block number for the start of the index and data sections of the file. Additionally, information is stored detailing the number of blocks for the index section; the number of blocks for the data section; the position within the file of the end of the header section; and the position within the file of the last data sample. The tag for this entry is '**blocks**', and is followed by the block information.

Bytes	Field Format	Value
1-2	Integer	Length of Entry (39)
3	Integer	Entry Type
4-11	Character	**blocks**
12-15	Integer	Block size (normally 2048 Bytes)
16-19	Integer	Number of blocks (inc. header, index and data)
20-23	Integer	File position pointer for end of header
24-27	Integer	Block number for start of index section
28-31	Integer	Number of blocks in the index section (norm. 1)
32-35	Integer	Block number for start of data section
36-39	Integer	File position pointer for last data entry

Number of Channels Entry

The third entry specifies the number of channels of data that are available within the file. The tag for this entry is '**nChans**', and is followed by the number of channels.

Bytes	Field Format	Value
1-2	Integer	Length of Entry (15)
3	Integer	Entry Type
4-11	Character	**nChans**
12-15	Integer	Number of channels

Time Key

The time for a sample of data is encoded in a data file, and is referred to as the time stamp ($Time_{stamp}$). The forth entry contains information that is used to decode $Time_{stamp}$ using the following relationship:

$$\mathbf{time = (Time_{stamp} - keyOffset)*timeScale - baseTime}$$

To minimise the size of a compressed time history file, data is normally stored for only those channels which have changed between sequential time intervals. However, data are also stored for all channels at a frequency indicated by information contained in the variable **fullInt**. The tag for this entry is '**timekey**', and is followed by the decoding information.

Bytes	Field Format	Value
1-2	Integer	Length of Entry (35)
3	Integer	Entry Type
4-11	Character	**timekey**
12-19	Floating Point	baseTime
20-27	Floating Point	timeScale
28-31	Integer	keyOffset
32-35	Integer	fullInt

Signal Names

The signal name for each channel of data stored in a file is documented in the fifth entry. The length of each signal name is limited to 16 characters, with the names stored sequentially. The tag for this entry is '**names**', and is followed by the signals names.

Bytes	Field Format	Value
1-2	Integer	Length of Entry (**11 + (16 x nChans)**)
3	Integer	Entry Type
4-11	Character	**names**
12-27	Character	Name of signal 1
28-43	Character	Name of signal 2
:	:	:
:	:	:

Optional Titles

Comments may be stored in the data file, but are limited in length to 1024 characters. The tag for each comment entry is '**titles**', and is followed by the comment character string.

Bytes	Field Format	Value
1-2	Integer	Length of Entry
3	Integer	Entry Type
4-11	Character	**titles**
12-?	Character	Character string for comment

Time Summary

This entry stores time information for the first data sample in a file and the last data sample. In addition, this entry contains a count of the total number of data samples stored in a file. The tag for this entry is **'times'**, and is followed by the time summary information.

Bytes	Field Format	Value
1-2	Integer	Length of Entry (**31**)
3	Integer	Entry Type
4-11	Character	**times**
12-19	Float	$Time_{stamp}$ for first data sample
20-27	Float	$Time_{stamp}$ for last data sample
28-31	Integer	Number of data samples in a file

End of Header

This is the final entry in the header section of a data file. It contains the tag **'endhead'**.

Bytes	Field Format	Value
1-2	Integer	Length of Entry (**11**)
3	Integer	Entry Type
4-11	Character	**endhead**

<u>Index Section</u>

This section contains information on the location of data within a file. The information is used to increase the speed for random access to data by providing the location of data in the file and the corresponding $Time_{stamp}$. $Time_{stamp}$ and data location are stored for each discrete time if there are less than 255 time samples in a file. Otherwise, $Time_{stamp}$ and data location corresponding to every i^{th} discrete time are stored, with i equal to the next whole number greater than the total number of time samples divided by 255.

Bytes	Field Format	Value
1	Integer	Number of entries in the index section
2-5	Integer	Index entry 1 $Time_{stamp}$
6-9	Integer	Index entry 1 data location
10-13	Integer	Index entry 2 $Time_{stamp}$
14-17	Integer	Index entry 2 data location
:	:	:
:	:	:

<u>Data Section</u>

The remainder of the file contains $Time_{stamp}$ information and data for each discrete time. The number of bytes of information stored for a discrete time is encoded in the two bytes preceding the $Time_{stamp}$. A single byte flag identifying the form used to store data follows the $Time_{stamp}$. The flag has a value of "0" for the last entry in the data section. The flag has a value of "1" when data for all channels is stored for a

discrete time, and the flag has a value of "2" when a partial list of data is stored. Information identifying the channel numbers of data stored for a discrete time will follow the flag if it has a value of "2". Data is encoded as 24 bit floating point numbers with each channel stored sequentially.

Bytes	Field Format	Value
1-2	Integer	Length of Entry
3-6	Integer	$Time_{stamp}$ for discrete time
7	Integer	Storage flag
8-10	Float	Data for channel
:	:	:
:	:	:

alternatively, for a storage flag = 2

Bytes	Field Format	Value
1-2	Integer	Length of Entry
3-6	Integer	$Time_{stamp}$ for discrete time
7	Integer	Storage flag
8-(8+n)	n x Byte	Channel information
(8+n)-(8+n+3)	Float	Data for channel
:	:	:
:	:	:

where **n** equals the next whole number greater than the total number of channels divided by 8.

Appendix C : Sample FORTRAN 77 Program

The examples used to illustrate the time history functions in sections three and four, have been presented using C programming language conventions. The functions may also be accessed from applications written other programming languages, for example FORTRAN 77. A sample FORTRAN 77 program which opens an existing time history file for reading and writes data to a second file, using the documented functions, is presented in this appendix.

The **os_comp.h** file contains identifiers denoting the development language and operating system. These are used by the time history functions to interface correctly with the application. A listing of the **os_comp.h** file for use with the sample FORTRAN 77 program is presented in this appendix.

Sample FORTRAN 77 Program

```
c*************************************************************************
c*  Test Program                                                         *
c*************************************************************************
c Variable Declaration required for reading a data file

      logical OpenR, OpenW, Fread, FSeek

      parameter (nSigs = 4, MAX_chan = 100)

      integer       j
      integer       nChans, Unit_no
      integer       Data_I(nSigs)
      real*8        time
      real*8        Data_R(MAX_chan)
      character*16  Chan_R(MAX_chan)
      character*16  SigsRead(nSigs)
      data Data_I/3,5,10,15/
      data SigsRead/'azagm', 'qm', 'alpham', 'thetam'/

      Unit_no = 1

c*************************************************************************

c Open an existing file for reading
      if (.not.OpenR(Unit_no, 'inFile.dat', nChans)) then
        write(6,*) 'Error Opening Read File'
      else

c Listing the signals available in a file
        call RSigs(Unit_no, Chan_R)
        do j = 1,nChans
          write(6,*) 'Signals :: ', Chan_R(j)
        enddo

c Selecting the signals to be read from the file by signal name
        call SigsR(Unit_no, SigsRead, nSigs)

c Selecting the signals to be read from the file by channel number
c        call ChansR(Unit_no, Data_I, nSigs)

        if (.not.OpenW(Unit_no, 'outFile.dat', nSigs, SigsRead, 'asc1'))
     .    then
          write(6,*) 'Error Opening Write File'
        else

c Reading nominated time points from the data file
          tSeek = 2.5
          if (.not.FSeek(Unit_no, tSeek, time, Data_R)) then
              write(6,*) 'Error reading data from getdata file'
          else
            write(6,*) 'tSeek, time :: ', tSeek, time
            do
              write(6,'(a8,g20.14)') 'Data :: ', Data_R(j)
            enddo
          endif
```

```
c Rewinding the file to the start of tha data block
c          call RewR(Unit_no);

c Reading the complete data file
100        if (FRead(Unit_no, time, Data_R)) then
             call FWrite(Unit_no, time, Data_R)
             do
               write(6,'(a8,g20.14)') 'Data :: ', Data_R(j)
             enddo
             goto 100
           endif

c End of OpenW Statement
         endif

c End of OpenR Statement
       endif

c Closing the Reading  and Writing Data files
       call CloseR(Unit_no)
       call CloseW(Unit_no)

c*******************************************************************************

       write(6,*) 'Program Completed'

c*******************************************************************************
```

OS_Comp.h File

```
/*******************************************************************************/
/* TimeHist :: os_comp.h                                                       */
/* Library of Routines to read and write time history data format files        */
/*                                                                             */
/* Application programming language and Operating System identifiers           */
/*******************************************************************************/

#ifndef __get_os_comp_h
#define __get_os_comp_h

/* Select the appropriate OS switch :: No selection implies PC - DOS :: Do Not Use
CMP3*/
/*#define __WIN32_OS*/  /* Compilation for WINDOWS OS */
#define __UNIX_OS       /* Compilation for UNIX OS */
/*#define __WIN16_OS*/  /* Compilation for WINDOWS 3.1* OS :: Do Not Use CMP3 */

/* Select the appropriate Machine Type ::  */
/*#define __I386*/
#define __RS6000
/*#define __PPC*/

/* Select the appropriate complitation language  :: Default C */
/*#define __C*/     /* Compilation for CPP programs */
/*#define __CPP*/   /* Compilation for CPP programs */
#define __FORTRAN   /* Compilation for FORTRAN programs */
#endif /* __get_os_comp_h */

/*******************************************************************************/
```

DISTRIBUTION LIST

Functions for Writing and Reading time History Data

Geoff Brian

AUSTRALIA

DEFENCE ORGANISATION

Task Sponsor
COPS HQAC

S&T Program
Chief Defence Scientist
FAS Science Policy } shared copy
AS Science Corporate Management
Director General Science Policy Development
Counsellor Defence Science, London (Doc Data Sheet)
Counsellor Defence Science, Washington (Doc Data Sheet)
Scientific Adviser to MRDC Thailand (Doc Data Sheet)
Director General Scientific Advisers and Trials/Scientific Adviser Policy and Command (shared copy)
Navy Scientific Adviser (Doc Data Sheet and distribution list only)
Scientific Adviser - Army (Doc Data Sheet and distribution list only)
Air Force Scientific Adviser
Director Trials

Aeronautical and Maritime Research Laboratory
Director
Chief of Air Operations Division
Research Leader – Avionics and Flight Mechanics
Research Leader – Air Operational Analysis
Research Leader – Simulation & Human Factors
Head, Air to Surface Operations

Author: Mr G.J. Brian

Mr J.S. Drobik
Mr B.A. Woodyatt
Mr A.D. Snowden
Mr K.L. Bramley

DSTO Library
Library Fishermens Bend
Library Maribyrnong

Library Salisbury (2 copies)
Australian Archives
Library, MOD, Pyrmont (Doc Data sheet only)

Capability Development Division
Director General Maritime Development (Doc Data Sheet only)
Director General Land Development (Doc Data Sheet only)
Director General C3I Development (Doc Data Sheet only)

Navy
SO (Science), Director of Naval Warfare, Maritime Headquarters Annex, Garden Island, NSW 2000 (Doc Data Sheet and distribution list only)

Army
ABCA Office, G-1-34, Russell Offices, Canberra (4 copies)
SO (Science), DJFHQ(L), MILPO Enoggera, Queensland 4051 (Doc Data Sheet only)
NAPOC QWG Engineer NBCD c/- DENGRS-A, HQ Engineer Centre Liverpool Military Area, NSW 2174 (Doc Data Sheet only)

Air Force
ARDU
STK2
ASCENG
Library

Intelligence Program
DGSTA Defence Intelligence Organisation
SO1-AW Defence Intelligence Organisation

Corporate Support Program (libraries)
OIC TRS, Defence Regional Library, Canberra
Officer in Charge, Document Exchange Centre (DEC), (Doc Data Sheet and distribution list only)
*US Defence Technical Information Center, 2 copies
*UK Defence Research Information Centre, 2 copies
*Canada Defence Scientific Information Service, 1 copy
*NZ Defence Information Centre, 1 copy
National Library of Australia, 1 copy

UNIVERSITIES AND COLLEGES
Australian Defence Force Academy
Library
Head of Aerospace and Mechanical Engineering

OTHER ORGANISATIONS

NASA (Canberra)
AGPS

OUTSIDE AUSTRALIA

ABSTRACTING AND INFORMATION ORGANISATIONS

INSPEC: Acquisitions Section Institution of Electrical Engineers
Library, Chemical Abstracts Reference Service
Engineering Societies Library, US
Materials Information, Cambridge Scientific Abstracts, US
Documents Librarian, The Center for Research Libraries, US

INFORMATION EXCHANGE AGREEMENT PARTNERS

Acquisitions Unit, Science Reference and Information Service, UK
Library - Exchange Desk, National Institute of Standards and Technology, US
National Aerospace Laboratory, Japan
National Aerospace Laboratory, Netherlands

SPARES (13 copies)

Total number of copies: **65**

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

Functions for Writing and Reading Time History Data

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

Geoff Brian

5. CORPORATE AUTHOR

Aeronautical and Maritime Research Laboratory
PO Box 4331
Melbourne Vic 3001

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-GD-0183	AR-010-554	General Document	June 1998

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
M1/9/336	A70-092	COPS HQAC	88	6

13. DOWNGRADING/DELIMITING INSTRUCTIONS

14. RELEASE AUTHORITY

Chief, Air Operations Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT

No limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

data transmission, functions, flight characteristics

19. ABSTRACT

The manipulation of time history data is one of the most common activities conducted when analysing aircraft flight behaviour and performance, and as a result there exists a multitude of data formats individualised for specific applications. Air Operations Division (AOD), of the Defence Science and Technology Organisation, have chosen to support two time history data formats defined by NASA Dryden Flight Research Center for development of flight behaviour and performance applications. A suite of functions for writing and reading the selected NASA time history data formats have been developed at AOD. These functions may be incorporated into analysis applications to reduce development time for new software, and to improve the sharing of data between applications.