AR-009-926

DSTO-TN-0064

A Database Compiler for Flight Dynamic Applications

S.D. Hill

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Mar-10

A Database Compiler for Flight Dynamic Applications

S. D. Hill

Aeronautical and Maritime Research Laboratory

DSTO-TN-0064

ABSTRACT

This report describes a database system developed by the Air Operations Division (AOD) for aircraft flight dynamic models. A new database format has been established to meet AOD's flight dynamic and performance requirements. A program has been written to create program source code to read and interpolate/extrapolate data stored in the database format. The major advantage of the new database format is that all the information required to read and interpolate/extrapolate a data set is contained within the database. A typical example of the database format, technical descriptions of the syntax, methods of interpolation/extrapolation, method of database interfacing, and procedure for database use are given.

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

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

Telephone: (03) 9626 7000
Fax: (03) 9626 7084

AR No. 009-926
December1996

APPROVED FOR PUBLIC RELEASE

Author

S. D. Hill

Air Operations Division

Shane graduated in 1987 with a Bachelor of Aeronautical Engineering (Honours) from the University of Sydney. He commenced work at the Aeronautical Research Laboratory (ARL) in 1988, working in the area of aircraft flight dynamics. He has developed a flight dynamic model for a joined wing aircraft configuration and a six degree of freedom flight dynamic model of a spinning aircraft. He has also been involved in the development of flight dynamic models for the F-111C and F/A-18 aircraft. Currently Shane is working on the reconstruction of F/A-18 flight paths based on data obtained from the maintenance data recorder fitted to the F/A-18 aircraft and a response following controller operating on a flight dynamic model of the aircraft. This process has been used to reconstruct the trajectory of a number of F/A-18 aircraft for RAAF incident inquiries.

A Database Compiler for Flight Dynamic Applications

EXECUTIVE SUMMARY

A database compiler has been developed by the Air Operations Division (AOD) for use in constructing aircraft flight dynamic and performance models and a database format has been established to meet the AOD requirements of both modelling activities. A program has been written to create program source code to read and interpolate/extrapolate data stored in this database format. The major advantage of this format is that all the information required to read and interpolate/extrapolate a data set is contained within the database. A typical example of the database format, technical descriptions of the syntax, method of interpolation/extrapolation, method of database interfacing, and procedure for database compiling is given.

The major benefit of the database compiler system within AOD is the introduction of a common database format and interpolation system. This format has been adopted by Flight Mechanics and Performance for all new software and any newly developed programs that require data interpolation can use this system. This commonality has greatly improved the rate at which flight dynamic and performance model databases may be produced, as the data access and interpolation/extrapolation routines have already been taken care of by the database compiler. The compiler also represents a faster and more efficient method of accessing the required databases.

The database compiler is currently being used by a variety of aircraft flight dynamic and performance models. The database compiler has been used to interpolate/extrapolate the aerodynamic data for a basic trainer spinning model, an F-111C flight dynamic model, and the performance characteristics of the F/A-18. The use of the database format and compiler is not confined to these specific examples drawn from Air Operations Division. It can be used by any application that requires the interpolation/extrapolation of multi-dimensional data.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS AND ACRONYMS

AMRL	Aeronautical & Maritime Research Laboratory
ANSI	American National Standards Institute
AOD	Air Operations Division
ASCII	American Standard Code for Information Interchange
dbc	Database Compiler
DSTO	Defence Science and Technology Organisation
NAWC-AD	Naval Air Warfare Center - Aircraft Division
RAAF	Royal Australian Air Force

TRADEMARKS

Borland C	Trademark of Borland International, Inc.
AIX, IBM PC	Trademark of International Business Machines
IRIX	Trademark of Silicon Graphics, Inc.
F77L	Trademark of Lahey Computer Systems, Inc.
MSDOS, Windows	Trademark of Microsoft Inc.
UNIX	Trademark of Bell Laboratories

1 INTRODUCTION

This report describes the format used in developing the aircraft flight dynamic model databases within the Air Operations Division (AOD) at the Aeronautical and Maritime Research Laboratory (AMRL) of the Defence Science and Technology Organisation (DSTO). A similar format was originally employed by the US Naval Air Warfare Center Aircraft Division (NAWC-AD) [1] for their F/A-18 flight dynamic model. This format has been extended and a compiler called **dbc** produced to create FORTRAN 77 [2] or C [3] programming language routines to read and interpolate/extrapolate data. The compiler has been built around an n-dimensional interpolation/extrapolation routine. The advantage of this particular database format is that all the information required to read and interpolate/extrapolate a data set is contained within the database itself. The database format uses ASCII character words that describe and direct the means by which data will be evaluated. This format has been used to develop databases for a flight dynamic model of a spinning aircraft based on a basic trainer [4], an update to the F-111C flight dynamic model [5], a flight dynamic model for the MK-82 store [6], and for an aircraft performance estimation utility [7] for the F/A-18 and F-111C. A typical example of the database format is given in Appendix A. This example is used in section 5 to describe the procedure for reading the database. A technical description of the syntax, method of interpolation/extrapolation, method of database interfacing, and procedure for compiling the database format is given in the following sections.

Note that this report describes the functionality of the database compiler **dbc** for version 2.6. The compiler is downwards compatible and will read **dbc** database formats of earlier versions.

2 DATABASE FORMAT AND COMPILER HISTORY

The database format and compiler described within this document was developed as part a flight dynamic model of a spinning aircraft based on basic trainer [4]. Large amounts of wind-tunnel data had to be read and interpolated to provide the aerodynamic force and moment information for the model's equations of motion. Aerodynamic data was available from three sources for the basic trainer; static wind-tunnel data, rotary balance wind-tunnel data, and data from a line vortex model. This data was to be formed into three separate databases.

A database format was required that was easy to interpret and generate. All database formats used within AOD for existing flight dynamic models were assessed. None of these database formats allowed the functionality of the data to be intuitively interpreted from the database source (i.e. the formats did not include information about how the data is to be read or manipulated). Evaluatingflight dynamic models acquired from outside DSTO, the F/A-18 flight dynamic model developed by NAWC-AD [1] had a format for its flight data database that was simple to read and understand. The basis of the format is the definition of functions that are to be interpolated. The functions define a set of independent variables and consequently the number of points the function holds for interpolation. This is the format that was adopted and improved upon and is described is section 3.

Once a database format was established, a program was required to read the data and provide interpolation routines to access the database. The database routine library had to be independent of the flight dynamic model so that it could be easily adapted to other flight dynamic models. The NAWC-AD F/A-18 flight dynamic model has database manipulating routines, however they were written using DEC VMS-specific functions. It was therefore decided to write database read and interpolation routines that were independent of operating system type and would ultimately be used with UNIX and DOS operating systems.

The first implementation of the database read and interpolation routines was a set of FORTRAN 77 routines that read the database at run-time and generated a table of user-defined dependent variables. This process was slow due to the interpretation of an ASCII file. Interpolation was carried out with several steps. The name of the user-defined dependent variable was passed to the interpolation routine. The dependent variable would then be searched for within the lookup table. The data would then be passed to the interpolation routine and a value returned. Several annoying problems were encountered with this library of routines:

- Interpolation was slowed by the need to look up dependent variable names within a table of user defined dependent variables.
- Database storage arrays were pre-allocated and were memory inefficient, i.e. the data were not dynamically assigned at run-time and data could not be unloaded from memory.
- Database storage arrays were defined in COMMON BLOCKS that the user had to edit to define data limits. This introduced problems due to the user not understanding the requirements of the COMMON BLOCKS.

The obvious solution was to write a program that would generate program source code specific to particular data. This eliminated all the problems above and allowed greater flexibility with the use of the database technique. Section 4 describes the use of the developed compiler. This compiler has been written to generate database read and interpolation routines for C and FORTRAN 77 programming languages for the UNIX, DOS, and Microsoft Windows operating environments.

3 DATABASE FORMAT DESCRIPTION

The database format consists of an ASCII character file of descriptive words that provide directives by which defined dependent variables are to be read and interpolated. An example database is given in Appendix A. The database format allows the definition of a dependent variable, say x, to be tabulated at discrete values of a number of independent variables $y_{i\ i=1,n}$. We wish to obtain the value of x at values of y_i not corresponding to those for which x is tabulated. That is we wish to evaluate

$$x = f(y_1, y_2, y_3, \ \cdots \ y_n) \tag{1}$$

at

$$y_1 = y_1', \quad y_2 = y_2', \quad y_3 = y_3', \quad \cdots \quad y_n = y_n'. \tag{2}$$

In the nomenclature of the database compiler, THRUPUT defines the names of the dependent and independent variables. CONBPT and VARBPT define the values of the independent variables y_i for which values of dependent variable x is tabulated. POINTS defines the values of the dependent variable x for all tabulated values of the independent variable y_i.

These descriptive words are termed *reserved*. That is to say they cannot be used as database variable names. The reserved words *are not case-sensitive* and are explained in detail in the following sections. Brief definitions are as follows:

TITLE: this directive defines the title for a specific data set.

THRUPUT: this directive defines a user-dependent variable as a function of independent variables for interpolation.

CONBPT: this directive defines values of the independent variables that have data values with a constant step.

VARBPT: this directive defines values of independent variables that have data values with a variable step.

POINTS: this directive defines the values of an independent variable for all tabulated values of the independent variables.

A single line in the ASCII file must not exceed 512 characters. Data past the 512th character will not be read. No warning is given if this occurs. Comments in the database file are denoted by special characters in the first column of the line. Special characters are considered to be any of the following:

```
~!@#$%^&*()<>{}[]|/\:;"'`,.?+-_=.
```

It is suggested, however, that for consistency only one should be used in a source database file. Preferred characters are `!`, `#`, or `%` since they are consistent with usage in many operating systems (e.g. UNIX).

The following conventions and symbols are used in the text to describe the form of the database file directives:

UPPER CASE	Upper case letters and words are to be written as shown, except where noted otherwise.
lower case	Lower case abbreviations and words represent characters or numerical values that are to be defined.
[ ]	Brackets are used to indicate optional items.
{ }	Braces surrounding two or more items indicate that at least one of the items must be specified.
|	The **or** symbol separates two or more optional items.
...	An ellipsis indicates that the preceding optional item(s) may appear more than once in succession.
( )	A pair of parentheses enclose entities that must be written as shown.

3.1 TITLE Directive

Function

Sets a title to classify the type of data to follow.

Syntax

`TITLE` **title-string**

Example

```
TITLE F-111C Longitudinal Flight Data
title Binary Shift Data
```

Method of Operation

The title **title-string** should be a general description of the data that are to follow. Any number of TITLE directives may be used in a file, including none. The title declared will be displayed to standard output when the database is being read into the user's program. There is no limit to the length of the title string except that defined by the maximum number of characters per line of database source (512 characters).

3.2 THRUPUT Directive

Function

Defines the name and functionality of a user defined dependent variable to be interpolated. All dependent and independent variables names are defined.

Syntax

THRUPUT **var(bpt [, bpt] ...),ext**

where:

var is the name of a dependent variable to be defined.

bpt is the name of a independent variable to be defined using the VARBPT or CONBPT directives. A **bpt** name of NULL has a special meaning and is described below.

ext is an extension string to append to the dependent variable name **var**.

Example

```
thruput Cma( alpha, alt, mach),flt
THRUPUT z( x, y)
```

Method of Operation

The THRUPUT directive is used to define the way in which a dependent variable is to be read and interpolated using a set of independent variables. The dependent variable name **var** must appear with a POINTS directive later in the database file and it will hold the values to be interpolated. The dimensionality of the dependent variable **var** is defined with independent variables **bpt**. Any number of independent variables can be defined. Each independent variable must appear with a CONBPT or VARBPT directive later in the database file and it will hold data defining the bounds of the interpolation. If no data is required for the independent variable, then the independent variable name must be NULL.

The NULL directive is used when a THRUPUT dependent-variable needs to appear as though it has higher dimensionality than the data indicate. It is possible that a program may require to allow the user to choose between several database sources, e.g. an F/A-18 or an F-111C database, as shown is section 4.7.2. The database would be easier to access if common THRUPUT dependent variables were accessed with the same arguments. For example, an F/A-18 database may have lift coefficient as a function of angle of attack, stabilator and rudder deflections, while the F-111C database may have lift coefficient as a function of the same independent variables as well as wing sweep. To make both aircrafts' lift coefficient variables have common calling arguments, the following THRUPUT definitions could be employed:

```
!     F-111C lift coefficient
      THRUPUT CL( AOA, STAB, RUDD, SWEEP),Flt

!     F/A-18 lift coefficient
      THRUPUT CL( AOA, STAB, RUDD, NULL),Flt
```

No sweep information need appear in the POINTS directive for the F/A-18 database. The dependent variable would be treated as a four-dimensional variable and not five-dimensional. The dependent variables would be accessed in the following manner.

```
!     F-111C CL_Flt call
      CL = CL_Flt( alpha, stab, rudder, sweep)

!     F/A-18 CL_Flt call
      CL = CL_Flt( alpha, stab, rudder, sweep)
```

For the F/A-18 routine call, the independent variable `sweep` would not be used in the interpolation of `CL_Flt`.

Method of Interface

The THRUPUT directive also defines the interface between the interpolation variable to the user source code. When the database is compiled, the name of the FORTRAN 77 or C function generated by the THRUPUT directive is made up from the dependent variable name **var** and its extension name **ext**. The C or FORTRAN 77 interpolation function name would have the form **var_ext**. By convention, extension is used to indicate the form of data being used. If the data were for example flight data, an extension of `flt` could be used, similarly an extension of `wt` or `cfd` could be used to denote wind-tunnel or computational fluid dynamic data respectively. If the extension name **ext** is not given, then the interpolation function name would be **var**. For example, the THRUPUT directives

```
THRUPUT Cma( alpha, alt, mach),flt
THRUPUT Cmq( alpha, alt, mach),wt
THRUPUT Cnr( alpha, alt, mach)
```

would generate a source code interface called

```
Cma_flt
Cmq_wt
Cnr
```

and would be accessed in a FORTRAN 77 program as

```
      real alpha, alt, mach, Cma, Cma_flt

      alpha = 10.0
      alt   = 25000.0
      mach  = 0.68
      Cma   = Cma_flt( alpha, alt, mach)
      write(6,*)'Cma = ', Cma
```

or in a C program as

```
#include "dbcinclude.h"
      float alpha, alt, mach, Cma;

      alpha = 10.0;
      alt   = 25000.0;
      mach  = 0.68;
      Cma   = Cma_flt( &alpha, &alt, &mach);
      printf( "Cma = %f\n", Cma);
```

Note that in the FORTRAN 77 code, the database interface function should be defined with a REAL statement. In C programs however, an **#include** statement is used to incorporate a file that contains all the function prototypes and defines for the database interfaces. This **include** file is created during the compilation of a database and will usually have a name of the form *databasesourcename.h*. See section 4.6 for more information on program coding.

3.3 CONBPT and VARBPT Directive

Function

Defines a data set to be used as an interpolation axis for a user-defined independent variable.

Syntax

bpt `CONBPT` **start end step**`/ [option] ...`
bpt `VARBPT` **start** `[` **data** `...]/ [option] ...`

where:

bpt is the independent variable name.

start is the start data point.

data is an intermediate data point.

end is the last data point.

step is the step size to use between start and end points to calculate intermediate points.

Example

```
alpha conbpt -5.0 25.0 2/ ubl=30.0 lbl=-8.5
beta  conbpt -10.0, 10.0, 5/ warning=on
x     VARBPT 0.0, 0.135, 0.275, 0.287, 0.568, 0.612,
             0.986/
```

Method of Operation

The CONBPT and VARBPT directives are used to define the values of each independent variable for which data are tabulated. Each independent variable name appearing in a CONBPT or VARBPT should have been previously defined via a THRUPUT directive or the defined independent variable will not be used. CONBPT is used where these values are uniformly spaced while VARBPT is used where their spacing is variable.

The values used for **start, end, data** and **step** may be either REAL/float or integer type. The last data value must be terminated with a forward slash (/).

3.3.1 Independent Variable Options

Several options can be used with the CONBPT and VARBPT directives to define the behaviour of the interpolation. These are:

LBL & UBL: specify the lower and upper bound limits respectively for the independent variable. If a value exceeds these limits, then the lower or upper bound limit value (see below) is passed back to the user code. If no LBL or UBL options are given, then the data set may be extrapolated to $\pm\infty$ (or the machine limit for these values).
Syntax: ubl={limit}.

LBV & UBV: specify the lower and upper bound values to return when a lower or upper bound limit is reached. If no LBV or UBV values are given, then the value returned is that interpolated/extrapolated by using the LBL or UBL values. If no LBL or UBL values are given then LBV and UBV are not processed.
Syntax: ubv={value}.

WARNING: specifies that warning messages should be given when a lower or upper bound limit is reached. The messages will appear on the standard output device, (normally the terminal screen). By default no warning messages are displayed.
Syntax: warning={on,off}.

3.4 POINTS Directive

Function

Defines the data set for a user defined dependent variable.

Syntax

var POINTS **data** [**data** ...]

where:

var is the name of the user defined dependent variable.

data are the values to be interpolated.

Example

```
Cla POINTS 0.00134 0.0056 0.0089 0.012 0.016 0.018/
z points 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0,
         9.0, 10.0/
```

Method of Operation

The directive POINTS defines the data values for a user defined dependent variable **var**. The POINTS directive must have appeared within a THRUPUT directive before being used. If it has not appeared previously in a THRUPUT statement, then the POINTS data is not processed and a warning message will be given by the **dbc**. The THRUPUT directive defines the functionality of dependent variable **var** and ultimately defines the number of data points that must be read by the POINTS directive. By default, data points are read with the independent variables set to their first defined values, then incremented through each independent variable range using the independent variable names as defined from right to left. That is, the order will be as follows for a generic THRUPUT variable:

$$\mathbf{var}(\mathbf{bpt_n}, ..., \mathbf{bpt_3}, \mathbf{bpt_2}, \mathbf{bpt_1}). \tag{3}$$

The last data value can be terminated with a forward slash (/), but is not necessary. For example, if a dependent variable Z has the following THRUPUT and VARBPT definitions:

```
thruput Z( X, Y)
X varbpt 0.0 2.0 4.0 5.0/
Y varbpt 0.0 6.0 13.0 17.0/
Z points 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0
         10.0 11.0 12.0 13.0 14.0 15.0/.
```

These values would be assigned to Z(X,Y) as follows:

1	Z(0.0, 0.0) = 0.0	9	Z(4.0, 0.0) = 8.0
2	Z(0.0, 6.0) = 1.0	10	Z(4.0, 6.0) = 9.0
3	Z(0.0, 13.0) = 2.0	11	Z(4.0, 13.0) = 10.0
4	Z(0.0, 17.0) = 3.0	12	Z(4.0, 17.0) = 11.0
5	Z(2.0, 0.0) = 4.0	13	Z(5.0, 0.0) = 12.0
6	Z(2.0, 6.0) = 5.0	14	Z(5.0, 6.0) = 13.0
7	Z(2.0, 13.0) = 6.0	15	Z(5.0, 13.0) = 14.0
8	Z(2.0, 17.0) = 7.0	16	Z(5.0, 17.0) = 15.0.

This order may be reversed. The option of data order is given because of the data order convention of FORTRAN and C being opposite to one another. The data order option is only a user personal issue and does not affect the functionality of the database and compiler. If reversed, the **-r** flag MUST be used when compiling the database with **dbc**. The THRUPUT variable will be processed in the following order:

$$\mathbf{var}(\mathbf{bpt_1}, \mathbf{bpt_2}, \mathbf{bpt_3}, ..., \mathbf{bpt_n}). \tag{4}$$

4 DATABASE COMPILER - dbc

A compiler has been written to generate either FORTRAN 77 or C source code routines to read and interpolate databases using the database format described in section 3. These routines are designed to be easily accessed from user code. The name of the database compiler is **dbc** and its function is described in the following sections.

4.1 Usage

```
dbc [-bC[F|W]dDruv -m[s|l] -o name] database_name ...
dbc -dT[F|C]
```

4.2 Description

The **dbc** command compiles **dbc** database format files into FORTRAN 77 or C source code. This source code allows the database to be read and interpolated from a user's own FORTRAN 77 or C programs with a minimum of interface requirements. The input database may have any file name. The use of a .db extension is suggested however.

4.3 Command Line Flag Options

-b Generate the binary database from the ASCII input database without creating any source code to read or interpolate the database. This is useful only when the data in the database has changed or multiple databases are to be read from one source file, i.e. the first database is compiled normally, then the remainder is compiled with the **-b** flag. This prevents previously generated **dbc** source code from being overwritten.

-C Translate database to C for C interfacing. Also creates a binary database file and libraries. The memory for the database will be dynamically allocated at run-time. Can be used with the -F flag. See -CF option.

-CF Translate database to C for FORTRAN 77 interfacing. This is the default translation method. Also creates a binary database file and libraries. The memory for the database will be dynamically allocated at run-time.

-CW Translate database to C for Microsoft Windows interfacing. Also creates a binary database file and libraries. The memory for the database will be dynamically allocated at run-time.

-d Make all interfacing routines have variables of double precision type.

-D Turn on debug mode. Debug mode produces an extended listing of information useful for tracking down problems during compilation of database files.

-F Translate database to FORTRAN 77 for FORTRAN 77 interfacing. The -CF option is more efficient for FORTRAN 77 code interfacing. Similar to the -C and -CF options, this option also creates a binary database file and libraries. However, libraries are still written in C because the binary database file is created in C binary. Can be used with the -C flag. See -CF option.

-T[F |C] Generate a FORTRAN 77 or C test program and database respectively. This test database can then be compiled with **dbc** to create the read and interpolation routines.

-m[s |l] Selects the type of memory model to use when generating the database read routines. The small memory model uses data statements in FORTRAN 77, or variable initialisation in C, to store the database values. This should only be used for small database files (one-dimensional and two-dimensional THRUPUT dependent variables only) and removes the overhead of reading data files at run-time. The large memory model creates a binary file from the ASCII input. This binary file is read when the database initialisation routine is called at run-time. A binary file is used for its compactness and speed of reading. The default is the large memory model.

-o name Name of the generated output source code file. If this option is not used, the name of the first database input file is used to create the name the output file.

-r Process THRUPUT independent variables from the left-most independent variable moving to the right. The default is to process from right to left. This option effects the order in which the POINTS data should be written to the ASCII database file. See section 3.4.

-u Include underscore extension to all C interfacing routines. This is used for FORTRAN 77 compilers that may append underscores to procedural names during compilation. Linking errors will occur if underscores are appended and this option is not used.

-v Verbose toggle. Default is on. Verbose provides more information during compilation of a database source file than would normally be given. Information about what files are being compiled, the names of libraries being created, and the way the user should compile the resultant source files is given. If turned off and there are no errors in the database file, then no information about the process will be seen on successful completion.

4.4 Compilation of dbc

The program **dbc** is written in ANSI C and will compile on most systems without problems. However, some system dependent definitions are required to overcome some differences in the way individual systems pass variables between routines, especially when mixing FORTRAN 77 and C code. The **dbc** source code allows the definition of operating system types. On compilation of **dbc**, at least one of the following system types must be defined.

AIX For IBM R6000 AIX machines using cc and xlf.

IRIX For Silicon Graphics IRIX machines using cc and f77.

MSDOS For IBM PC or Microsoft Windows using Borland C and Lahey F77L. The FORTRAN 77 translation will only work correctly with Lahey F77L.

For example, to compile the **dbc** source code for AIX, the following command should be used:

```
cc -O dbc.c -DAIX -o dbc
```

4.5 Interpolation/Extrapolation Algorithm

The interpolation/extrapolation algorithm employed in the database compiler is based on a linear method. It is able to interpolate in an arbitrary number of dimensions so long as the data for the dimensions exist. At present a limit of fifteen dimensions has been placed on the compiler to prevent possible over-run of memory. A defined THRUPUT dependent variable of fifteen

dimensions would take a large amount of data to define. For example, if a fifteen-dimensional dependent variable were defined with independent variables that had five data points per dimension, then there would be approximately 3.05×10^{10} points to define in the POINTS directive. Thus far, the maximum dimension defined in an actual application by a user has been seven.

To explain how the linear interpolation scheme works, consider figure 1 which sets out the terminology for a one dimensional interpolation. Let x be the value found by interpolation; x_1 and

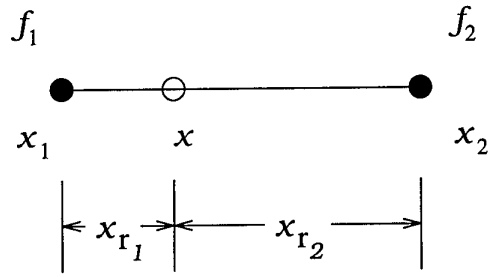

Figure 1: One-dimensional Interpolation

x_2 are the closest values to x. Let f_1 and f_2 be the values returned by $f(x_1)$ and $f(x_2)$ respectively. Let x_{r_1} and x_{r_2} be the ratios of the distances between x and x_1, x and x_2 as follows:

$$\begin{aligned} x_{r_1} &= \frac{x - x_1}{x_2 - x_1} \\ x_{r_2} &= 1 - x_{r_1}. \end{aligned} \tag{5}$$

Then $f(x)$ is given by the equation:

$$f(x) = x_{r_2} f_1 + x_{r_1} f_2. \tag{6}$$

This example is fairly trivial, but increasing the example to a two-dimensional interpolation adds complexity. The two-dimensional situation is illustrated in figure 2. Let x and y be the values required to interpolate $f(x, y)$. Let x_1, y_1, x_2, and y_2 be the closest values to x and y. Let f_1, f_2, f_3, and f_4 be the values returned by $f(x_1, y_1)$, $f(x_2, y_1)$, $f(x_1, y_2)$, and $f(x_2, y_2)$ respectively. Let x_{r_1} and x_{r_2} be the ratios of the distance between x and x_1, x and x_2 as shown in equation 5. Also let y_{r_2} and y_{r_2} be the ratios of the distance between y and y_1, y and y_2 as follows:

$$\begin{aligned} y_{r_1} &= \frac{y - y_1}{y_2 - y_1} \\ y_{r_2} &= 1 - y_{r_1} \, etc. \end{aligned} \tag{7}$$

To interpolate for $f(x, y)$, the intermediate values a and b need to be known and are evaluated as follows:

$$\begin{aligned} a = f(x, y_1) &= x_{r_2} f_1 + x_{r_1} f_2 \\ b = f(x, y_2) &= x_{r_2} f_3 + x_{r_1} f_4. \end{aligned} \tag{8}$$

The point $f(x, y)$ can now be evaluated as follows:

$$\begin{aligned} f(x, y) &= y_{r_2} a + y_{r_1} b \\ &= y_{r_2}(x_{r_2} f_1 + x_{r_1} f_2) + y_{r_1}(x_{r_2} f_3 + x_{r_1} f_4) \end{aligned}$$

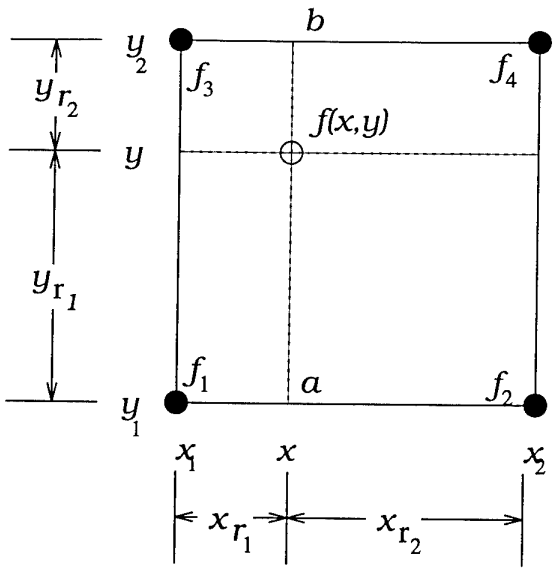

Figure 2: Two-dimensional Interpolation

$$
\begin{aligned}
&= y_{r_2}x_{r_2}f_1 + y_{r_2}x_{r_1}f_2 + y_{r_1}x_{r_2}f_3 + y_{r_1}x_{r_1}f_4 \\
&= f_1\prod_{j=1}^{2} f_{r_{1j}} + f_2\prod_{j=1}^{2} f_{r_{2j}} + f_3\prod_{j=1}^{2} f_{r_{3j}} + f_4\prod_{j=1}^{2} f_{r_{4j}} \\
&= \sum_{i=1}^{4}(f_i\prod_{j=1}^{2} f_{r_{ij}}) \qquad (9)
\end{aligned}
$$

where $f_{r_{ij}}$ denotes the corresponding axis ratio for the j^{th} dimension at the i^{th} node (a node being one of the edge data points to be interpolated). Extending equation 9 to n-dimensions produces:

$$f(x,y) = \sum_{i=1}^{2^n}(f_i\prod_{j=1}^{n} f_{r_{ij}}) \qquad (10)$$

where n is the dimensionality of the function to interpolate and 2^n is the number of closest bounds to sum with their corresponding axis ratios.

4.6 Interfacing dbc Generated Code with User Code

To assist in learning how to interface user code with the database read and interpolation routines, **dbc** may be used to produce a test database and test programs for FORTRAN 77 or C with the commands `dbc -TF` and `dbc -TC` respectively. The test database that is created illustrates how the database may be used to convert a binary number into a decimal number and is called *dbcsample.dat*. The aim of the test database is to illustrate most of the features that can be included. It also provides a platform on which users can quickly build their own code. The test database is:

```
        TITLE   Binary to decimal converter
*
```

```
      THRUPUT BinToDec( bit1, bit2, bit3, bit1, bit1, bit1),tst
*
bit1   VARBPT 0,1/ lbl = 0 ubl = 1
bit2   VARBPT 0,1/ lbl = 0 lbv = -10.0 ubl = 1 ubv = 10.0 warn=on
bit3   CONBPT 0,1,1/
*
BinToDec POINTS
   0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
   21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,
   41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,
   61,62,63/
```

This test database defines one six-dimensional dependent variable `BinToDec`. Most of the input fields to `BinToDec` are a function of independent variable `bit1` which has values 0 and 1 as given by the independent variable directive VARBPT. The upper bound limit directive states that if `bit1` ever exceeds a value of 1, then that value will be replaced with 1. Likewise the lower bound limit directive states that if `bit1` ever falls below a value of 0, then that value will be replaced with 0. This means that `bit1` will never be extrapolated past these two boundaries.

The independent variables `bit2` and `bit3` are given for example only and are not required in the conversion from binary into decimal. `bit2` shows how the upper and lower bound value directive operates. If `bit2` exceeds the upper bound value of 1, then the returned value from the THRUPUT dependent variable `BinToDec` will be 10.0. Likewise if the lower bound limit of 0 is reached, a value of -10.0 is returned. Note also that the warning directive has been turned on with `warn=on`. This means that a warning message will be displayed whenever the upper or lower bound limits are reached.

The independent variable `bit3` is defined with the constant independent variable directive CONBPT. Although not required, it states that `bit1` will have values 0 through 1 in steps of 1 (i.e. only values 0 and 1). No extra directives have been placed on `bit3` which indicates that values entered at this level may be extrapolated to infinity.

All the data for `BinToDec` are listed after the POINTS directive. Here the data are ordered such that the rightmost independent variable in the THRUPUT directive is varied most quickly. A flag option exists in the **dbc** compiler if the reverse order is required.

To compile this test database one of the following commands would be used, depending on whether a FORTRAN 77 or C interface is required:

```
dbc -F dbcsample.dat     ! FORTRAN 77 for FORTRAN 77
dbc -CF dbcsample.dat    ! C for FORTRAN 77 interface
dbc -C dbcsample.dat     ! C for C interface
dbc -CW dbcsample.dat    ! C for Microsoft Windows interface.
```

The first command will generate FORTRAN 77 read and interpolation routines for use with FORTRAN 77 source code. This command, although still available, has been superseded largely by the second command that creates C read and interpolation routines for a FORTRAN 77 interface. The reason is that the C routines can manage the database memory requirements better through dynamic allocation. This allows database files to be loaded and unloaded as required. The last two commands generate routines in C for C user source code.

After compilation a binary form of the database will be generated and called *dbcsample.bin*. The database initialisation and `BinToDec` interface routines will be written either to *dbcsample.f* or *dbcsample.c*. The initialisation routine has the following naming convention:

```
init_db_<input_database_name>
```

The <input_database_name> is taken from the **dbc** command line and will not include the file extension. Therefore, in this example the initialisation routine will be called *init_db_dbcsample*. The initialisation routine is responsible for reading the binary database file and manipulating the data ready for use by the interfacing routines. In this example, only one THRUPUT dependent variable was defined and the interface to BinToDec will be called *BinToDec_tst*. The extension of *_tst* was defined at the end of the THRUPUT directive.

To read this database, two test programs may be generated called *dbctest.f* and *dbctest.c*. The FORTRAN 77 and C test programs are:

```
C
C       DBC FORTRAN test program
C
        program dbctest
        real f, r1, r2, r3, r4, r5, r6

        call init_db_dbcsample
        do while ( .TRUE.)
          write( 6, '(a,$)') 'Enter six bits (e.g. 0 1 0 0 0 1) :: '
          read( 5, *) r1, r2, r3, r4, r5, r6
          f = BinToDec_tst( r1, r2, r3, r4, r5, r6)
          write( 6, *) 'Decimal      = ', f
          write( 6, *) 'Return bits = ', r1, r2, r3, r4, r5, r6
          write( 6, *)
        enddo
        end
```

```
/*
   DBC C test program
 */
#include <stdio.h>
#include "dbcsample.h"

void main( void)
{
  float f, r1, r2, r3, r4, r5, r6;

  init_db_dbcsample();
  while ( 1) {
    printf( "Enter six bits (e.g. 0 1 0 0 0 1) :: ");
    scanf( "%f %f %f %f %f %f", &r1, &r2, &r3, &r4, &r5, &r6);
    f = BinToDec_tst( &r1, &r2, &r3, &r4, &r5, &r6);
    printf( "Decimal      = %f\n", f);
    printf( "Return bits = %f %f %f %f %f %f\n\n", r1, r2, r3, r4, r5, r6);
  }
}
```

In the test programs, only two procedural calls are used to interface with the database routines. The first is *init_db_dbcsample* which reads the database into memory. The second is *BinToDec_tst* which is the routine that interfaces to the interpolation routine. Note that in the C interface these values must be passed by pointer so that if any upper or lower bound limit directives are used, the return values can be passed back to the user code. Table 1 gives the result of input to this sample program.

Input						Result	Comment
0	1	0	0	0	1	17	Normal conversion
1	0	1	0	1	0	42	"
1	1	1	1	1	1	63	"
0	1.1	0	0	0	1	10	Test of upper bound limit on `bit2`
1	0	1000	0	0	1	8033	Test of 'infinite' extrapolation on `bit3`

Table 1: **dbc** Sample Test Program Results

4.7 Multiple Database Access

The database compiler **dbc** has been written to allow easy access to multiple databases by one of two methods. The first uses a nested **IF** block structure to select between the different databases. The second uses a technique of making the THRUPUT section of each database common, thus requiring only one interpolation structure.

4.7.1 Nested IF block method

This method compiles each required database and generates unique initialisation routines for each. To access a database, the initialisation routine for that database is called once only at the beginning of the program, then the required THRUPUT dependent variable defined is accessed using the interface routine. For example, for the following database files:

```
f111c.db, f18.db
```

the following database initialisation routines would be created with **dbc**:

```
init_db_f111c()
init_db_f18().
```

Each database in this case must have unique THRUPUT dependent variable names otherwise the compiler will not be able to determine which variable should be called. The user then must create the required code structure to select and interface between the four databases. This means that the F-111C and F/A-18 databases would have a lift coefficient interface routine as follows:

```
...
if ( F111) then
  call init_db_f111c()
  CL = CL_F111( ALT, MACH, DELHT, ALPHA, SWEEP)

else if ( F18) then
  call init_db_f18()
  CL = CL_F18( ALT, MACH, ALPHA, STAB)

endif
...
```

4.7.2 Common THRUPUT Method

The second method can only be used with the C libraries for FORTRAN 77 or C interfaces only. This method accesses multiple databases through a single initialisation routine and definition of interpolation variables using one interface structure. That is, rather than using *CL_F111* and *CL_F18* as shown in the previous example, a single name of *CL_FLT* could be used. This is achieved by making the databases have a common THRUPUT section by looking for similarities in the THRUPUT dependent variables required for interpolation. For example, the previous example has one thing in common; they are all aircraft databases. Therefore it is logical to assume that the THRUPUT dependent variables required to define the aircrafts' characteristics would be similar. The F-111C database has a THRUPUT definition for lift, drag and pitching moment coefficients as follows:

```
THRUPUT CL( ALT, MACH, DELHT, ALPHA, SWEEP),FLT
THRUPUT CDLB( MACH, SWEEP),FLT
THRUPUT CM( ALT, MACH, DELHT, ALPHA, SWEEP),FLT
```

The F/A-18 aircraft has the following THRUPUT structure for the same dependent variables:

```
THRUPUT CL( ALT, MACH, ALPHA, STAB),FLT
THRUPUT CDLB( MACH),FLT
THRUPUT CM( ALT, MACH, ALPHA, STAB),FLT
```

The main differences between these two structures is that the F-111C aircraft has an extra parameter of SWEEP; and the stabilator and angle of attack independent variables (DELHT, STAB, ALPHA), are reversed. These two structures can be made equivalent by firstly swapping the ALPHA and DELHT independent variables in the F-111C aircraft database (making sure that this is reflected in the POINTS directive for all independent variables swapped), as follows:

```
THRUPUT CL( ALT, MACH, ALPHA, DELHT, SWEEP),FLT
THRUPUT CDLB( MACH, SWEEP),FLT
THRUPUT CM( ALT, MACH, ALPHA, DELHT, SWEEP),FLT
```

The F/A-18 THRUPUT structure must have the same dimensionality as the F-111C THRUPUT structure. This can be achieved with the NULL independent variable directive in the following manner:

```
THRUPUT CL( ALT, MACH, ALPHA, STAB, NULL),FLT
THRUPUT CDLB( MACH, NULL),FLT
THRUPUT CM( ALT, MACH, ALPHA, STAB, NULL),FLT
```

The NULL independent variable directive has no size and has no effect on the data defined in the POINTS directive. It merely serves to add an extra dimension to the interfacing routines at the location it is defined.

Once all required database files have the same THRUPUT structure, one must be compiled to generate the initialisation and interface routines. All other database files can be compiled to generate the binary data file only and they are read at run time. If the F-111C database file is compiled, it will generate an initialisation file called *init_db_f111c*. This routine will be structured as follows:

```
void init_db_f111c( void)
{
  int  nbp, ntp;
  FILE *fp;
```

```
  if ( ver_notdone) {
    printf( "Database Compiler V-2.6\n");
    ver_notdone = FALSE;
  }
  printf( "Initialising \"f111c\"\n");
  printf( "  :- F111C Database Version 1.0\n");
  if (( fp = fopen( "f111c.bin", "rb")) == NULL)
    error_msg( 2, "f111c.bin", NULL);
  if ( fread( &nbp, sizeof( int), 1, fp) != 1)
    error_msg( 4, "cl_flt", (char *) NULL);
  if ( fread( &ntp, sizeof( int), 1, fp) != 1)
    error_msg( 5, "cl_flt", (char *) NULL);
  if ( fread( levels_cl_flt, sizeof( int), 5, fp) != 5)
    error_msg( 6, "levels", "cl_flt");
  if ( bp_cl_flt != NULL)
    free( bp_cl_flt);
  if (( bp_cl_flt = (float *) calloc( nbp, sizeof( float))) == NULL)
    error_msg( 7, "breakpoint", "cl");
  if ( fread( bp_cl_flt, sizeof( float), nbp, fp) != nbp)
    error_msg( 1, "breakpoint", "cl");
  if ( tp_cl_flt != NULL)
    free( tp_cl_flt);
  if (( tp_cl_flt = (float *) calloc( ntp, sizeof( float))) == NULL)
    error_msg( 7, "thruput", "cl");
  if ( fread( tp_cl_flt, sizeof( float), ntp, fp) != ntp)
    error_msg( 1, "thruput", "cl'");
  .
  .
  .
     More THRUPUT variables to be read.
  .
  .
  .
  if ( fclose( fp) != 0)
    error_msg( 3, "f111c.bin", NULL);
}
```

To make this a generic routine able to read all the aircraft databases in the example, the initialisation routine would be renamed to something like *init_db*. All references to *f111c* or *f111c.bin* must be replaced with a C character pointer *dbname*. The pointer *dbname* must be declared as an argument to *init_db* by replacing *(void)* with *char *dbname*. The new initialisation routine would then be structured similar to the following code:

```
void init_db( char *dbname)
{
  int  nbp, ntp;
  FILE *fp;

  if ( ver_notdone) {
    printf( "Database Compiler V-2.6\n");
    ver_notdone = FALSE;
  }
  printf( "Initialising \"%s\"\n", dbname);
  if (( fp = fopen( dbname, "rb")) == NULL)
    error_msg( 2, dbname, NULL);
  if ( fread( &nbp, sizeof( int), 1, fp) != 1)
    error_msg( 4, "cl_flt", (char *) NULL);
  if ( fread( &ntp, sizeof( int), 1, fp) != 1)
    error_msg( 5, "cl_flt", (char *) NULL);
  if ( fread( levels_cl_flt, sizeof( int), 5, fp) != 5)
    error_msg( 6, "levels", "cl_flt");
```

```
  if ( bp_cl_flt != NULL)
    free( bp_cl_flt);
  if (( bp_cl_flt = (float *) calloc( nbp, sizeof( float))) == NULL)
    error_msg( 7, "breakpoint", "cl");
  if ( fread( bp_cl_flt, sizeof( float), nbp, fp) != nbp)
    error_msg( 1, "breakpoint", "cl");
  if ( tp_cl_flt != NULL)
    free( tp_cl_flt);
  if (( tp_cl_flt = (float *) calloc( ntp, sizeof( float))) == NULL)
    error_msg( 7, "thruput", "cl");
  if ( fread( tp_cl_flt, sizeof( float), ntp, fp) != ntp)
    error_msg( 1, "thruput", "cl");
  .
  .
  .
     More THRUPUT variables to be read.
  .
  .
  .
  if ( fclose( fp) != 0)
    error_msg( 3, dbname, NULL);
}
```

User code can now access the required aircraft database using the following FORTRAN 77 code:

```
      character*128 dbname
      character*1   NULL

      NULL = char( 0)
      write(6,'(a,$)') 'Enter database name (f111c/f18) :: '
      read(5,'(q,a)') lengthdbn, dbname

      call init_db( dbname( 1:lengthdbn) // '.bin' // NULL)
      cl = cl_flt( alt, mach, alpha, stab, sweep)
```

The same sequence would be used with C code. However the appending of a NULL character would not be necessary. It is only required when passing character strings from FORTRAN 77 to C routines. This method completely removes the need for the complex IF block structure shown in the first method.

5 EXAMPLE - F-111C DATABASE

Appendix A gives an example of part of the database format used for the F-111C flight dynamic model. Each line of the database example will be explained. The line

```
TITLE  F-111C MODEL DATABASE LONGITUDINAL V-1.1 trial only 4/10/1993
```

is setting the title to the database and has no influence over the method for reading the database. The first of the THRUPUT commands is:

```
THRUPUT CL(SWEEPB,ALT2,MACHH,DELHT2,ALPHAB)
```

and defines a dependent variable called CL (lift coefficient) which is a function of aircraft sweep (SWEEPB), altitude (ALT2), mach number (MACHH), horizontal stabilator deflection (DELHT2),

and angle of attack (ALPHAB). SWEEPB, ALT2, MACHH, DELHT2, and ALPHAB are known as independent variables. These variables define the functionality of dependent variable CL. The number of points required to be read at the POINTS directive is defined with either the CONBPT or VARBPT directive. The independent variable definition for SWEEPB is given by the line:

```
SWEEPB VARBPT 16.0,26.0,35.0,45.0,50.0,
              72.5/LBL=16.0,UBL=72.5
```

This indicates that the dependent variable CL has database values for sweep angles of 16°, 26°, 35°, 45°, 50°, and 72.5° and therefore has six values on this axis. The options LBL and UBL state that the database for sweep should not be extrapolated below 16° or above 72.5° respectively. Similarly, CL has values for altitude at 5, 10, 20, 30, and 40 thousand feet given by the VARBPT directive as follows:

```
ALT2   VARBPT 5000.0,10000.0,20000.0,30000.0,
              40000.0/LBL=5000.0,UBL=40000.0
```

The same method for reading can be applied to independent variables MACHH and DELHT2 which have thirteen and five values for their respective axes. Independent variable ALPHAB however shows the use of a CONBPT directive as follows:

```
ALPHAB CONBPT -4.0,20.0,2.0/LBL=-4.0,UBL=20.0
```

The dependent variable CL will therefore have values for angle of attacks of -4° through 20° in 2° increments to give thirteen values on this axis. Multiplying the number of values for each of the independent variable axes together will give the number of data points that need to be read at the POINTS directive for CL. The number of data points to be read for CL is 25 350, calculated as follows:

SWEEPB		ALT2		MACHH		DELHT2		ALPHAB		
6	×	5	×	13	×	5	×	13	=	25 350

The values for CL are read with the POINTS directive and are given as:

```
CL     POINTS
              -0.760058999,  -0.563299000,  -0.366538972,  -0.169778988,
               0.026981026,   0.223740995,   0.394610941,   0.498090923,
               0.562229216,   0.595377624,   0.595377624,   0.595377624,
                    "
                    "
                    " Partial data shown only.
                    "
                    "
               0.074630000,   0.178690001,   0.282750010,   0.386810005,
               0.490870029,   0.594930053,   0.698989987,   0.803050041,
               0.887478352,   0.952275097
```

The order in which these data points are allocated for CL is as follows. First, all independent variables SWEEPB, ALT2, MACHH, DELHT2, and ALPHAB are set to their first defined values. Subsequent points are read by incrementing through the rightmost defined independent variable (ALPHAB) until its last value is reached. The next independent variable to the left

(DELHT2) is incremented by one value and the right most independent (ALPHAB) set back to its first. The right most independent variable (ALPHAB) is again incremented to its last value. This system continues until all independent variables have been processed through their ranges. This is illustrated in the following table.

Point	SWEEPB	ALT2	MACHH	DELHT2	ALPHAB	Value
1	16.0	5000.0	0.0	-25.0	-4.0	-0.760058999
2	16.0	5000.0	0.0	-25.0	-2.0	-0.563299000
3	16.0	5000.0	0.0	-25.0	0.0	-0.366538972
4	16.0	5000.0	0.0	-25.0	2.0	-0.169778988
5	16.0	5000.0	0.0	-25.0	4.0	0.026981026
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
25346	72.5	40000.0	1.8	10.0	12.0	0.594930053
25347	72.5	40000.0	1.8	10.0	14.0	0.698989987
25348	72.5	40000.0	1.8	10.0	16.0	0.803050041
25349	72.5	40000.0	1.8	10.0	18.0	0.887478352
25350	72.5	40000.0	1.8	10.0	20.0	0.952275097

6 CONCLUSION

A database format and compiler used in the development of aircraft flight dynamic model databases within AOD at DSTO-AMRL has been presented. Information on both the syntax and method of reading the database format has been included. A comprehensive explanation of the interpolation technique used by the database compiler has also been presented. An example F-111C database file has been used to illustrate the techniques involved.

ACKNOWLEDGEMENTS

The author would like to thank Andrew Snowden (of DSTO AMRL AOD), for his time and effort in helping the author get this document through the publication process. Without Andrews expertise on "walking" a document through the vetting and publication process, this document may not have been published.

REFERENCES

[1] Nichols, J. H., *"Control Analysis and Simulation Test Loop Environment (CASTLE) User's Guide CASTLE Version 3.0"*, Strike Aircraft Test Directorate, Naval Air Warfare Center, Patuxent River, Maryland 20670-5304, January 1993.

[2] Didday, R. and Page, R., *"FORTRAN for Humans"*, West Publishing Company, Minnesota, USA, 1981.

[3] Kernighan, B. W. and Ritchie, D. M. *"The C Programming Language"*, AT&T Bell Laboratories, Prentice Hall, New Jersey, USA, 1978.

[4] Hill, S. D. and Martin, C. A., *"A Flight Dynamic Model of Aircraft Spinning"*, Flight Mechanics Report 180, AR-005-600, Defence Science and Technology Organisation Aeronautical Research Laboratory, Melbourne, Australia, June 1990.

[5] Cooper, M. I., *"A Flight Dynamic Model of the F111-C using the Simulation Language ACSL"* (CONFIDENTIAL), Aerodynamics Report 166, AR-004-067, Defence Science and Technology Organisation Aeronautical Research Laboratory, Melbourne, Australia, December 1985.

[6] Vergos, A., *"Further Development of a Generalised Six Degree of Freedom Flight Dynamic Model of an Unguided Store"*, Wackett Centre Report No. CR 93/12, Sir Lawrence Wackett Centre for Aerospace Design Technology, Royal Melbourne Institute of Technology, Australia, December 1993.

[7] MacPherson, B. P., *"Development of a Graphical User Interface for a Suite of Aircraft Performance Estimation Programs"*, Wackett Centre Report No CR 93/10, Sir Lawrence Wackett Centre for Aerospace Design Technology, Royal Melbourne Institute of Technology, Australia, October 1993.

Appendix A - F-111C Database Format Example

```
*
       TITLE  F-111C MODEL DATABASE LONGITUDINAL V-1.1 trial only 4/10/1993
*
       THRUPUT CL(SWEEPB,ALT2,MACHH,DELHT2,ALPHAB)
       THRUPUT CM(SWEEPB,ALT2,MACHH,DELHT2,ALPHAB)
       THRUPUT TR(SWEEPA,ALT1,MACHA,ALPHAB,DELHT1)
       THRUPUT CDMIN(SWEEPA,MACHA)
       THRUPUT CLMIN(SWEEPA,MACHA)
       THRUPUT CLBR(SWEEPA,MACHA)
       THRUPUT CDLB(SWEEPA,MACHA)
       THRUPUT CDLS(SWEEPA,MACHA)
       THRUPUT CMQ(SWEEPB,MACHH,ALPHAB)
       THRUPUT CMAD(SWEEPB,MACHH,ALPHAB)
       THRUPUT CLQ(SWEEPB,ALPHAB,MACHH)
       THRUPUT CLAD(SWEEPB,MACHH,ALPHAB,ALT3)
       THRUPUT CLAH(MACHF)
*
ALPHAB CONBPT -4.0,20.0,2.0/LBL=-4.0,UBL=20.0
SWEEPA VARBPT 16.0,26.0,35.0,50.0,72.5/LBL=16.0,UBL=72.5
SWEEPB VARBPT 16.0,26.0,35.0,45.0,50.0,72.5/LBL=16.0,UBL=72.5
MACHA  VARBPT .0,.4,.6,.7,.8,.85,.9,.95,1.0,
              1.2,1.58,1.6,2.0,2.07,2.2,2.5/LBL=0.0,UBL=2.5
MACHF  VARBPT 0.0,0.2,0.4,0.6,0.8,0.9,0.95,1.0,
              1.05,1.1,1.2,1.6,2.0,2.5/LBL=0.0,UBL=2.5
MACHH  VARBPT 0.0,0.4,0.5,0.6,0.7,0.8,0.9,1.0,
              1.1,1.2,1.4,1.5,1.8/LBL=0.0,UBL=1.8
ALT1   VARBPT 0.0,15000.0,35000.0,50000.0,60000.0/LBL=0.0,UBL=60000.0
ALT2   VARBPT 5000.0,10000.0,20000.0,30000.0,40000.0/LBL=5000.0,UBL=40000.0
ALT3   VARBPT 0.0,5000.0,10000.0,20000.0,30000.0,40000.0/LBL=0.0,UBL=40000.0
DELHT1 VARBPT -25.0,-10.0,-4.0,0.0,4.0,10.0/LBL=-25.0,UBL=10.0
DELHT2 VARBPT -25.0,-10.0,0.0,4.0,10.0/LBL=-25.0,UBL=10.0
*
CL     POINTS
                -0.760058999,  -0.563299000,  -0.366538972,  -0.169778988,
                 0.026981026,   0.223740995,   0.394610941,   0.498090923,
                 0.562229216,   0.595377624,   0.595377624,   0.595377624,
                      "
                      "
                      " Partial data shown only.
                      "
                      "
                 0.074630000,   0.178690001,   0.282750010,   0.386810005,
                 0.490870029,   0.594930053,   0.698989987,   0.803050041,
                 0.887478352,   0.952275097
CM     POINTS
                 0.697396994,   0.709969878,   0.722542644,   0.735115528,
                 0.747688353,   0.760261118,   0.773426414,   0.787884593,
                 0.791329622,   0.779234648,   0.747074664,   0.690322995,
                      "
                      "
                      " Partial data shown only.
                      "
                      "
                -0.091829039,  -0.156257093,  -0.229905427,  -0.304699570,
                -0.380075365,  -0.455472916,  -0.530332088,  -0.604088902,
                -0.676181197,  -0.747301221
```

A Database Compiler for Flight Dynamic Applications
S. D. Hill and A. D. Snowden

AUSTRALIA

1. DEFENCE ORGANISATION

S&T Program

Chief Defence Scientist
FAS Science Policy
AS Science Industry and External Relations
AS Science Corporate Management
} shared copy

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 (3 copies Doc Data Sheet and one copy of the distribution list)
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
Head, Air to Surface operations
Head, Counter Air operations
Head, AOSC
Head, Flight Mechanics
Head, Helicopter Operations
Head, Maritime Patrol and Training
Head, Operations and Performance Analysis
Head, Simulation Application
Head, Simulation Technology
Head, Transport and Logistics Operations
Author: S. D. Hill
J. S. Drobik
A. D. Snowden
G. J. Brian
B. A. Woodyatt
S. B. Astill
K. L. Bramley
G. McKenzie
R. Caldeira

Electronics and Surveillance Research Laboratory

Director

DSTO Library

Library Fishermens Bend
Library Maribyrnong
Library DSTOS (2 copies)
Australian Archives
Library, MOD, Pyrmont (Doc Data Sheet)

Forces Executive

Director General Force Development (Sea), (Doc Data Sheet)
Director General Force Development (Land), (Doc Data Sheet)
Director General Force Development (Air)

Army

ABCA Office, G-1-34, Russell Offices, Canberra (4 copies)

Air Force

Aircraft Research and Development Unit
Tech Reports, CO Engineering Squadron, ARDU
LSA-DTA
ATS-3

S&I Program

Defence Intelligence Organisation
Library, Defence Signals Directorate (Doc Data Sheet only)

B&M Program (libraries)

OIC TRS, Defence Central Library
Officer in Charge, Document Exchange Centre (DEC), 1 copy
DEC requires the following copies of public release reports to meet exchange agreements under their management:
*US Defence Technical Information Centre, 2 copies
*UK Defence Research Information Center, 2 copies
*Canada Defence Scientific Information Service, 1 copy
*NZ Defence Information Centre, 1 copy
National Library of Australia, 1 copy

2. **UNIVERSITIES AND COLLEGES**

Australian Defence Force Academy
Library
Head of Aerospace and Mechanical Engineering
Deakin University, Serials Section (M list)
Deakin University Library, Senior Librarian
Monash University, Hargrave Library
Flinders University, Library
RMIT
Library
Aerospace Engineering
UNSW
Aeronautical Engineering
University of Newcastle
Library
Institute of Aviation

3. **OTHER ORGANISATIONS**

NASA (Canberra)
AGPS

OUTSIDE AUSTRALIA

4. ABSTRACTING AND INFORMATION ORGANISATIONS

Engineering Societies Library, US
Documents Librarian, The Center for Research Libraries, US

5. INFORMATION EXCHANGE AGREEMENT PARTNERS

Acquisitions Unit, Science Reference and Information Service, UK
National Aerospace Laboratory, Japan (RAAF sponsored reports)
National Aerospace Laboratory, Netherlands (RAAF sponsored reports)

6. UNITED STATES OF AMERICA

Department of the Navy
Naval Air Warfare Center Aircraft Division, Patuxent River, Maryland
Mr T. R. Fitzgerald
Mr B. W. York

SPARES (10 copies)

Total number of copies: 78

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)
UNCLASSIFIED

2. TITLE

A Database Compiler for Flight Dynamic Applications

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)

S D. Hill

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-TN-0064	AR-009-926	Technical Note	December 1996

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
M1/9/37	AIR 94/205	DTA	38	7

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

Database
Compiler
Flight Dynamics
F/A-18 Aircraft

19. ABSTRACT

This report describes a database system developed by the Air Operations Division (AOD) for aircraft flight dynamic models. A new database format has been established to meet AOD's flight dynamic and performance requirements. A program has been written to create program source code to read and interpolate/extrapolate data stored in the database format. The major advantage of the new database format is that all the information required to read and interpolate/extrapolate a data set is contained within the database. A typical example of the database format, technical descriptions of the syntax, methods of interpolation/extrapolation, method of database interfacing and procedure for database use are given.