Australian Government
Department of Defence
Defence Science and
Technology Organisation

Vector and Matrix Variable Definitions in DAVE-ML

Geoff Brian

Air Vehicles Division
Defence Science and Technology Organisation

DSTO-TN-1146

ABSTRACT

The Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML) is a syntactical language for exchanging flight vehicle dynamic model data. It has been developed in conjunction with the ANSI/AIAA S-119-2011 Flight Dynamics Model Exchange Standard prepared by the American Institute of Astronautics and Aeronautics (AIAA) Modeling and Simulation Technical Committee (MSTC). The purpose of DAVE-ML is to provide a framework to encode entire flight vehicle simulation data packages for exchange between simulation applications and the long-term archiving of model data. This report documents an extension to DAVE-ML for managing data as vectors or n-dimensional matrices. The attributes and dependencies of the additional DAVE-ML elements are discussed, and examples of encapsulating data as vectors and matrices are provided.

Published by

Air Vehicles Division
DSTO Defence Science and Technology Organisation
506 Lorimer St
Fishermans Bend, Victoria 3207 Australia

Telephone: (03) 9626 7000
Fax: (03) 9626 7999

AR-015-431
December 2012

APPROVED FOR PUBLIC RELEASE

Vector and Matrix Variable Definitions in DAVE-ML

Executive Summary

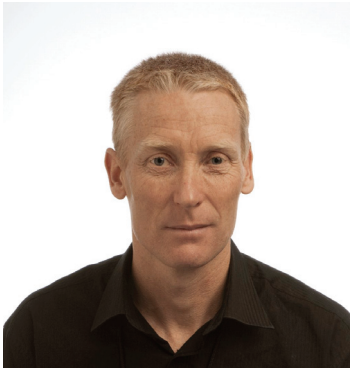
The Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML) is a syntactical language for exchanging flight vehicle dynamic model data. It has been developed in conjunction with the ANSI/ AIAA S-119-2011 Flight Dynamics Model Exchange Standard, prepared by the American Institute of Astronautics and Aeronautics (AIAA) Modeling and Simulation Technical Committee (MSTC). The intended purpose of DAVE-ML is to provide a framework for encoding entire flight vehicle simulation data packages for exchange between simulation applications and the long-term archiving of model data. Such data packages are commonly used in research, engineering development, and flight training simulations. DAVE-ML is designed to provide a programming-language-independent representation of aerospace vehicle characteristics, such as the aerodynamics, mass, propulsion, navigation and control properties.

Version 2.0.1 of DAVE-ML supports the storage of static aerospace vehicle characteristic data in human-readable text form, together with directives for automating the conversion of data into a form suitable for use in vehicle simulations. It provides a capability to include statistical properties for embedded data, such as confidence bounds and uncertainty ranges; along with references to reports, contact information and data provenance. However, additional functionality is required to support the exchange of entire flight vehicle simulation models. This includes supporting vector and matrix data, abstracting sub-system models, detailing dynamics system models (both discrete and continuous), and defining a dynamic data format (such as time-history data) for validating dynamics system models and the overall simulation package.

This report documents a DAVE-ML consistent syntax for managing data as vectors and n-dimensional matrices, including syntax for computing vectors and matrices using equations, and examples of encoding data using the syntax. The syntax offers an approach to improve the clarity of encoding aerospace vehicle dynamic model data, and thereby enhance the ability of DAVE-ML to encode entire flight vehicle dynamic simulation models and their validation data. This, in turn, will simplify the exchange of vehicle model data between simulation applications.

Author

Mr. Geoff Brian
Air Vehicles Division



Geoff Brian commenced employment at the DSTO in 1989, after graduating from the University of New South Wales with a degree in Aeronautical Engineering. During his employment with DSTO, he has been involved with the analysis of flight data used to develop a comprehensive flight dynamic model of the F-111C aircraft; the development of aircraft performance estimation applications; and coordination of DSTO support to a project replacing the F-111C aircraft's Pratt & Whitney TF30-P-103 engines with 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. During 1998 and 1999 Mr. Brian attended Loughborough University in the United Kingdom being awarded with a MSc in Industrial Mathematical Modelling. From 2000 - 2005 he coordinated DSTO support to the KC-30 Aerial Tanker acquisition program. More recently he contributed to and coordinated the development of an aircraft flight dynamic and simulation capability within Air Vehicles Division, and contributed to uninhabited aircraft systems flight dynamic research at DSTO. In 2010 Mr Brian undertook a Defence Science Fellowship at NASA Langley researching aircraft aerodynamic systems identification and vehicle modelling and simulation practices.

Contents

Abbreviations

AIAA	American Institute of Aeronautics and Astronautics
DAVE-ML	Dynamic Aerospace Vehicle Exchange Markup Language
DSTO	Defence Science and Technology Organisation
DTD	Document Type Definition
MathML	Mathematical Markup Language
NASA LaRC	National Aeronautics and Space Administration, Langley Research Center
XML	eXtensible Markup Language
W3C	World Wide Web Consortium

Nomenclature

$\boldsymbol{C}$, $\boldsymbol{C_i}$	Coefficient vector or component
$\boldsymbol{F}$, $\boldsymbol{F_i}$	Force vector or component
$\boldsymbol{g}$	Gravitational acceleration
$\boldsymbol{I}$, $\boldsymbol{I_{ii}}$	Mass-moment-of-inertia tensor or component
$\boldsymbol{m}$	Vehicle mass
p	Vehicle body-axes angular roll rate
q	Vehicle body-axes angular pitch rate
$\bar{\boldsymbol{q}}$	Kinematic pressure, $\bar{\boldsymbol{q}} = 0.5\boldsymbol{\rho V}^2$
$\boldsymbol{r}$	Vehicle body-axes angular yaw rate
$\boldsymbol{S}$	Reference area
$\boldsymbol{u}$	Vehicle body-axes axial velocity
$\dot{\boldsymbol{u}}$	Vehicle body-axes axial acceleration
v	Vehicle body-axes transverse velocity
$\dot{\boldsymbol{v}}$	Vehicle body-axes transverse acceleration
$\boldsymbol{V}$	Vehicle body-axes velocity vector , V = [u,v,w]
$\dot{\boldsymbol{V}}$	Vehicle body-axes acceleration vector , $\dot{\boldsymbol{V}} = [\dot{\boldsymbol{u}}, \dot{\boldsymbol{v}}, \dot{\boldsymbol{w}}]$
$\boldsymbol{w}$	Vehicle body-axes normal/vertical velocity
$\dot{\boldsymbol{w}}$	Vehicle body-axes normal/vertical acceleration
$\boldsymbol{\phi}$	Euler roll angle
$\boldsymbol{\theta}$	Euler pitch angle
$\boldsymbol{\rho}$	Atmospheric density
$\boldsymbol{\Omega}$	Body-axes rotational rate vector

[T]	Transformation matrix

Superscripts/Subscripts

be	Body to earth axes transformation
aerodynamic	Aerodynamic
propulsion	Propulsion
x	Vehicle body-axes x direction
y	Vehicle body-axes y direction
z	Vehicle body-axes z direction

Acknowledgements

The work documented in this report was conducted while on attachment to NASA Langley Research Center under a Visiting Researcher Agreement facilitated through the National Institute of Aerospace, Hampton, Virginia, USA. I wish to acknowledge the all staff at the National Institute of Aerospace who assisted with this attachment.

I wish to also acknowledge the valuable comments provided in correspondence with Dennis Linse, Bruce Jackson (NASA LaRC), Shane Hill (DSTO), Dan Newman (Quantitative Aeronautics) and Rob Curtin (Qinetiq Consulting Australia) during the formulation of the concepts that are encompassed in this document.

1. Introduction

The Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML) is a syntactical language for exchanging flight vehicle dynamic model data [1]. It was developed in conjunction with the ANSI/AIAA S-119-2011 Flight Dynamics Model Exchange Standard [2], prepared by the American Institute of Astronautics and Aeronautics (AIAA) Modeling and Simulation Technical Committee. The intended purpose of DAVE-ML is to provide a framework for encoding entire flight vehicle dynamic model data packages for exchange between simulation applications and the long-term archiving of model data. Such data packages are commonly used in research, engineering development, and flight training simulations.

DAVE-ML provides a programming-language-independent representation of aerospace vehicle characteristics, such as the aerodynamics, mass, propulsion, navigation and control properties. It employs a text-based format built upon the eXtensible Markup Language (XML) Version 1.1 [3], and the Mathematical Markup Language (MathML) Version 2.0 [4] open standards developed by the World Wide Web Consortium (W3C). DAVE-ML defines additional grammar to provide a domain-specific language for aerospace flight dynamics modelling, verification, and documentation. Version 2.0.1 of DAVE-ML, [5], is able to store static aerospace vehicle characteristic data in human-readable text form, together with directives for automating the conversion of data into a form suitable for use in vehicle simulations. Furthermore, it provides the capability to include statistical properties for embedded data such as confidence bounds and uncertainty ranges, along with references to reports, contact information and data provenance.

While Version 2.0.1 of DAVE-ML provides much of the functionality envisioned for exchanging aerospace vehicle data, it only supports scalar time-independent data. Additional functionality is required to support the exchange of entire flight vehicle dynamic models. This includes supporting vector and matrix data, abstracting sub-system models, detailing dynamics system models (both discrete and continuous), and defining a dynamic data format (such as time-history data) to support validation of dynamics system models and the overall simulation package. This report documents extensions to DAVE-ML to permit the management of vector and matrix data. Additional syntax for storing data as vectors or matrices is defined, together with syntax for computing vectors and matrices using an equation. Examples of encoding data using the extended syntax are also provided.

The vector and matrix syntax improves DAVE-ML's ability to encode entire flight vehicle dynamic simulation models, and in-turn the exchange of aerospace vehicle model data.

2. Simulation Modelling using Vectors and Matrices

Collating data as vectors and matrices permits parameters that have a common basis to be grouped when exchanging vehicle data, and in doing so, improve data clarity. For example, an aircraft's three body-axes aerodynamic force components (F_x, F_y, and F_z), or force coefficient components (C_x, C_y, and C_z), may be managed as a single vector parameter $F_{aerodynamic}$ as shown in Equation (1), where $\bar{q}$ is the kinematic pressure and S the reference area.

$$F_{aerodynamic} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_{aerodynamic} = \bar{q}\,S \begin{bmatrix} C_x \\ C_y \\ C_z \end{bmatrix}_{aerodynamic} \tag{1}$$

Similarly, the aircraft's mass-moments-of-inertia may be grouped as an inertia tensor I, Equation (2).

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \tag{2}$$

Furthermore, collating parameters into vector and matrices permits aerospace flight behaviour simulation applications to use generalised equations for defining the vehicle's motion instead of the more commonly used component forms of these equations. This has the added benefit of simplifying the coding of simulation applications. For example, Equations (3-5) represent the flat non-rotating earth body-axes linear accelerations of a vehicle commonly used in simulation applications, while Equation (6) represents the generalised equation for the vehicle acceleration.

$$\dot{u} = \frac{1}{m}\left(\bar{q}\,S\,C_x + F_{x\,propulsion}\right) - qw + rv + g\sin\theta \tag{3}$$

$$\dot{v} = \frac{1}{m}\left(\bar{q}\,S\,C_y + F_{y\,propulsion}\right) - ru + pw + g\sin\phi\cos\theta \tag{4}$$

$$\dot{w} = \frac{1}{m}\left(\bar{q}\,S\,C_z + F_{z\,propulsion}\right) - pv + qu + g\cos\phi\cos\theta \tag{5}$$

$$\dot{V} = \frac{1}{m}\left(\bar{q}\,S\,C_{aerodynamic} + F_{propulsion}\right) - \Omega \times V + g[T]^{be} \tag{6}$$

In Equation (6), $C_{aerodynamic}$ represents the aerodynamic force coefficient vector, $F_{propulsion}$ the propulsion force vector, Ω the rotational rate vector (p,q,r), V the linear velocity vector (u,v,w), and $[T]$ the earth-axes to body-axes Euler transform matrix.

3. Vector and Matrix Syntax Elements

Vectors and matrices are analogous to scalar variables except that they represent a series of values instead of a singular value. The capability of the DAVE-ML variable definition element, **variableDef,** has been extended to embrace the encoding of data as scalars, vectors or matrices. Sub-elements are included to define the dimensions of vectors and matrices, and to specify the data of the vector or matrix. These data can be numeric, references to other defined variables, or a combination. Alternatively, the contents of a vector or matrix can be computed using the **calculation** sub-element by defining a suitable MathML expression. Figure 1 represents the revised variable definition for DAVE-ML where the additional sub-elements supporting vector and matrix definitions, highlighted in bold text, are listed in Table 1.

```
variableDef : name, varID, units, [axisSystem], sign, alias,
              symbol, [initialValue], minValue, maxValue

  description?
  (provenance | provenanceRef)?
  (dimensionDef | dimensionRef)?
  (calculation | array)?
  (isInput | isControl | isDisturbance)?
  IsState?
  IsStateDeriv?
  IsOutput?
  IsStdAIAA?
  uncertainty?
```

Figure 1: DAVE-ML variable definition with additional sub-elements supporting vectors and matrices

Table 1: Additional sub-elements for the DAVE-ML variable definition supporting vectors and matrices

array	The data for the vector or matrix
dimensionDef	A list of dimensions (dim) for the vector or matrix
dimensionRef	A reference to a dimensionDef element

Figure 2 illustrates an example defining a vector using this syntax, while examples of defining two- and three-dimensional matrices are presented in Figure 3.

```
<variableDef name="vectorName_nd" varID="vectorID" units="nd">
  <dimensionDef dimID="vector_3">
    <dim>3</dim> <!-- Number of data points in the vector -->
  </dimensionDef>
  <array>
    <dataTable> 1.0, 2.0, 3.0 </dataTable>
  </array>
</variableDef>
```

Figure 2: An example of defining a vector using the extended DAVE-ML variable definition syntax

```
<variableDef name="matrixName_nd" varID="matrixID" units="nd">
  <dimensionDef dimID="matrix_2x3">
    <dim>2</dim> <!-- Number of rows in the matrix -->
    <dim>3</dim> <!-- Number of columns in the matrix -->
  </dimensionDef>
  <array>
    <dataTable>
      1.0, 2.0, 3.0, <!-- Row #1 -->
      0.0, 2.0, 5.0, <!-- Row #2 -->
    </dataTable>
  </array>
</variableDef>
```

a) A two-dimensional matrix,

```
<variableDef name="matrixName_nd" varID="matrixID" units="nd">
  <dimensionDef dimID="matrix_2x2x3">
    <dim>2</dim> <!-- Number of 2x3 matrices -->
    <dim>2</dim> <!-- Number of rows in the matrix -->
    <dim>3</dim> <!-- Number of columns in the matrix -->
  </dimensionDef>
  <array>
    <dataTable>
      <!-- Matrix #1 -->
      1.0, 2.0, 3.0, <!-- Row #1 -->
      0.0, 2.0, 5.0, <!-- Row #2 -->
      <!-- Matrix #2 -->
      7.0, 8.0, 9.0, <!-- Row #1 -->
      2.0, 9.0, 6.0, <!-- Row #2 -->
    </dataTable>
  </array>
</variableDef>
```

b) A three-dimensional matrix,

Figure 3: Example of defining two- and three-dimensional matrices using the extended DAVE-ML variable definition syntax

The list of dimensions for a vector or matrix is defined using the **dimensionDef** element, specifying either the number of data entries in a vector or the size of each dimension in an n-dimensional matrix. The lowest entry of the list specifies the number of columns of the base matrix. The subsequent higher entries specify the number of rows in the base matrix, the number of base matrices making up the third dimension, and so forth for n-dimensional matrices. This is illustrated in Figure 4.

```
<dimensionDef dimID="vector_3">
  <dim>3</dim> <!-- Number of entries in the vector -->
</dimensionDef>
```

a) A vector with three entries,

```
<dimensionDef dimID="matrix_2x2x2">
  <dim>2</dim> <!-- Number of matrices for the 3rd dimension -->
  <dim>2</dim> <!-- Number of rows in the base matrix -->
  <dim>2</dim> <!-- Number of columns in the base matrix -->
</dimensionDef>
```

b) A three-dimensional matrix,

```
<dimensionDef dimID="matrix_n_dimension">
  <dim>2</dim> <!-- Number of matrices for the nth dimension -->
  ..........
  <dim>2</dim> <!-- Number of matrices for the 3rd dimension -->
  <dim>2</dim> <!-- Number of rows in the base matrix -->
  <dim>2</dim> <!-- Number of columns in the base matrix -->
</dimensionDef>
```

c) An n-dimensional matrix,

Figure 4: Examples of defining the dimensions of vectors and matrices using the extended DAVE-ML variable definition syntax

A reference to a dimension definition can be used as an alternative to explicitly specifying the dimensions as part of the variable definition. This permits the reuse of **dimensionDef** elements for vectors or matrices having a common size. The **dimensionRef** references an identifier **dimID** associated with the dimension definition, as illustrated in Figure 5.

```
<dimensionDef dimID="matrix_2x3">
  <dim>2</dim> <!-- Number of rows in the matrix -->
  <dim>3</dim> <!-- Number of columns in the matrix -->
</dimensionDef>

<variableDef name="matrixName_nd" varID="matrixID" units="nd">
  <dimensionRef dimID="matrix_2x3">
  <array>
    <dataTable>
      1.0, 2.0, 3.0, <!-- Row #1 -->
      0.0, 2.0, 5.0, <!-- Row #2 -->
    </dataTable>
  </array>
</variableDef>
```

Figure 5: An example of using a reference to define the dimensions of a two-dimensional matrix

The data for a vector or matrix are specified using the **array** element. This is in effect a table of data, and thus the **dataTable** element from DAVE-ML is utilised for encoding the vector and matrix entries. The table of data represents consecutive entries for a vector. The entries for a matrix are specified such that the column entries of the first row are listed first followed by column entries for subsequent rows until the base matrix is complete. This sequence is repeated for higher order matrix dimensions until all entries of the matrix are specified. Figure 6 illustrates this procedure.

```
<array>
  <dataTable>
    0.0, eulerInclinationAngle, eulerRollAngle
  </dataTable>
</array>
```

a) A vector with three entries,

```
<array>
  <dataTable>
    <!-- First 2x2 Matrix -->
    1.0, 0.0, <!-- Row #1 -->
    0.0, 1.0, <!-- Row #2 -->

    <!-- Second 2x2 Matrix -->
    inertiaIXX, -20.4,      <!-- Row #1 -->
    -15.6     , inertiaIYY, <!-- Row #2  -->
  </dataTable>
</array>
```

b) A three-dimensional matrix,

Figure 6: An example of encoding data as either a vector or matrix using the extended DAVE-ML variable definition syntax

3.1 Supplementary Elements

The **dimensionDef**, **dimensionRef** and **array** elements are dependent on sub-elements to encode vector and matrix data. The **dimensionDef** uses a sub-element of **dim** to define the size of each dimension of the vector or matrix. An optional attribute **dimID** can also be specified defining a reference identifier permitting the instance of the dimension definition to be reused. The **dimensionRef** specifies the reference identifier of an associated **dimensionDef**, while **array** uses the **dataTable** sub-element to encode the entries of the vector or matrix. It is therefore necessary to include definitions for these elements within the DAVE-ML syntax. Figure 7 presents the additional syntax for DAVE-ML defining the supplementary element supporting vector and matrix data.

```
<!ELEMENT dimensionDef (dim+)>
<!ATTLIST dimensionDef dimID ID #IMPLIED>
```

a) The definition of the *dimensionDef* element added to the DAVE-ML syntax

```
<!ELEMENT dim (#PCDATA)>
```

b) The definition of the *dim* element

```
<!ELEMENT dimensionRef EMPTY>
<!ATTLIST dimensionRef dimID IDREF #REQUIRED>
```

c) The definition of the *dimensionRef* element

```
<!ELEMENT array (dataTable)>
```

d) The definition of the *array* element

Figure 7: Supplementary element definitions for encoding data as vectors and matrices using the extended DAVE-ML syntax

4. Computing Vectors and Matrices Using MathML

The **calculation** sub-element of the DAVE-ML variable definition permits the value of a variable to be computed from an equation defined using the open standard MathML syntax, [4]. MathML contains operators for computing the transpose, inverse, determinant, vectorproduct[1], scalarproduct[2], and outerproduct[3] of vectors and matrices. These operators

[1] The MathML definition for **vectorproduct** is equivalent to that of the **cross product** [5, 6].

are in addition to operators for adding, subtracting, and multiplying data. Furthermore, MathML contains an operator, named **selector**, which identifies entries to be extracted from a vector or matrix. Operators such as **plus**, **times**, and **minus** apply equally to scalars, vectors and matrices, and can be used when mixing variable types. The **transpose**, **inverse**, **determinant**, **vectorproduct**, **scalarproduct**, **outerproduct**, and **selector** operators are only relevant to vector and matrix variable types.

Utilising an equation to compute variables involving vectors and matrices is similar to computing scalar variables through the equivalent DAVE-ML constructs. However, it is necessary to define the size of the vector or matrix that results from the calculation. This is not required if the result of the calculation is a scalar value. An example of multiplying a matrix **M** and a vector **V** to calculate a resultant vector **R**, as shown in Equation (5), is presented in Figure 8.

$$R = [M]V \qquad \begin{bmatrix} \square \\ \square \end{bmatrix} = \begin{bmatrix} \square & \square & \square \\ \square & \square & \square \end{bmatrix} \begin{bmatrix} \square \\ \square \\ \square \end{bmatrix} \tag{5}$$

```
<variableDef name="R" varID="R" units="">
  <description> Multiplying a matrix and vector</description>
  <dimensionDef>
    <dim>2</dim> <!-- Number of entries in the output vector-->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <times/>
        <ci>M</ci>
        <ci>V</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

Figure 8: An example of multiplying a matrix by a vector.

A more complex example is presented in Figure 9, where Equation (6) is encoded using MathML vector and matrix operators.

[2] The MathML definition for **scalarproduct** is equivalent to that of the **dot product** [5, 6].
[3] The **outerproduct** multiplies two vectors to form a matrix: $A = uv^T$, where u and v are vectors [4].

$$R = [I]^{-1}\{\bar{T}(r \cdot r) \times Q\} \tag{6}$$

$$\begin{bmatrix} \square \\ \square \\ \square \end{bmatrix} = \begin{bmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{bmatrix} \left\{ \sqrt{\begin{bmatrix} \square & \square & \square \end{bmatrix}} \left(\begin{bmatrix} \square \\ \square \end{bmatrix} \cdot \begin{bmatrix} \square \\ \square \end{bmatrix} \right) \times \begin{bmatrix} \square \\ \square \\ \square \end{bmatrix} \right\}$$

```
<variableDef name="R" varID="R" units="">
  <description>
    Vector and Matrix operations using MathML
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of entries in the output vector -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <times/>
        <apply>
          <inverse/>
          <ci>I</ci>
        </apply>
        <apply>
          <vectorproduct/>
          <apply>
            <times/>
            <apply>
              <transpose/>
              <ci>T</ci>
            </apply>
            <apply>
              <scalarproduct/>
              <ci>r</ci>
              <ci>r</ci>
            </apply>
          </apply>
          <ci>Q</ci>
        </apply>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

Figure 9: An example of coding vector and matrix algebra using the MathML operators.

Sections 4.1 to 4.7 detail the MathML vector and matrix operators and their use in defining vector and matrix variables through the DAVE-ML syntax. In addition, examples are presented for each operator. Appendix A presents more complex examples of the vector and matrix syntax.

4.1 Transpose

The operation of transposing a vector or matrix is defined using the **transpose** operator. It applies only to one vector or matrix, which is identified using a reference to a predefined variable definition. The syntax for the transpose operator is:

```
<transpose/>
<ci>vectorOrMatrix</ci>
```

Figure 10 illustrates transposing a [2,3] matrix to form a [3,2] matrix.

```
<variableDef name="matrixTranspose"
             varID="matrixTranspose" units="">
  <description>
    This represents an example of the transposing a 2x3 matrix.
    matrix1: [2,3];
    result = ~matrix1: [3,2];
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of rows in the matrix -->
    <dim>2</dim> <!-- Number of columns in the matrix -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <transpose/>
        <ci>matrix1</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

Figure 10: An example of transposing a 2x3 matrix

4.2 Inverse

The operation of computing the inverse of a matrix is defined using the **inverse** operator. MathML also uses this operator to specify the inverse of a function. When used for matrix operations it applies only to one matrix, which is square (i.e. n x n). The matrix is identified using a reference to a predefined variable definition. The syntax for the inverse operator is:

```
<inverse/>
<ci>matrix</ci>
```

Figure 11 illustrates inverting a [3,3] matrix.

```
<variableDef name="matrixInverse"
             varID="matrixInverse" units="">
  <description>
    This represents an example of inverting a 3x3 matrix.
    matrix1: [3,3];
    result = !matrix1: [3,3];
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of rows in the matrix -->
    <dim>3</dim> <!-- Number of columns in the matrix -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <inverse/>
        <ci>matrix1</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

Figure 11: An example of inverting a 3x3 matrix

4.3 Determinant

The operation of computing the determinant of a matrix is defined using the **determinant** operator. It is a requirement that the matrix must be square (i.e. n x n) to compute the determinant, where the result is a scalar. It applied only to one matrix, which is identified using a reference to a predefined variable definition. The syntax for the determinant operator is:

```
<determinant/>
<ci>matrix</ci>
```

Figure 12 illustrates the procedure for computing the determinant of a matrix.

```
<variableDef name="matrixDeterminant"
             varID="matrixDeterminant" units="">
  <description>
    This represents an example of computing the determinant of
    a 3x3 matrix.
    matrix1: [3,3];
    result = |matrix1|: [scalar];
  </description>
  <calculation>
    <math>
      <apply>
        <determinant/>
        <ci>matrix1</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

Figure 12: An example of computing the determinant of a 3x3 matrix

4.4 Scalarproduct

The operation of computing the scalar-product of two vectors is defined using the **scalarproduct** operator. The scalar-product is equivalent to the *dot-product* as defined in References [5] and [6]. The two vectors are required to have the same number of entries, and are identified using references to predefined variable definitions. The result is a scalar. The syntax for the **scalarproduct** operator is:

```
<scalarproduct/>
<ci>vector1</ci>
<ci>vector2</ci>
```

Figure 13 illustrates the procedure for computing the scalar-product of two vectors.

```
<variableDef name="scalarProduct"
             varID="scalarProduct" units="">
  <description>
    This represents an example of computing the scalar-product
    of two vectors – vector1 and vector2.
    result = vector1.vector2: [scalar];
  </description>
  <calculation>
    <math>
      <apply>
        <scalarproduct/>
        <ci>vector1</ci>
        <ci>vector2</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

Figure 13: An example of computing the scalar-product of two vectors

4.5 Vectorproduct

The operation of computing the vector-product of two vectors is defined using the **vectorproduct** operator. The vector-product is equivalent to the *cross-product* as defined in References [5] and [6]. The two vectors are each required to have three entries, and they are identified using references to predefined variable definitions. The result is a vector of three entries. The syntax for the **vectorproduct** operator is:

```
<vectorproduct/>
<ci>vector1</ci>
<ci>vector2</ci>
```

Figure 14 illustrates the procedure for computing the vector-product of two vectors.

```
<variableDef name="vectorProduct"
             varID="vectorProduct" units="">
  <description>
    This represents an example of computing the vector-product
    of two vectors with 3 entries each – vector1 and vector2.
    result = vector1 x vector2: [3];
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of entries in the result vector -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <vectorproduct/>
        <ci>vector1</ci>
        <ci>vector2</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

Figure 14: An example of computing the vector-product of two vectors

4.6 Outerproduct

The operation of computing the outer-product of two vectors is defined using the **outerproduct** operator. The outer-product multiplies two vectors to form a matrix: $A = uv^T$, where u and v are vectors, [4]. The size of the resulting matrix is:

number of rows = number of entries of u, and
number of columns = number of entries of v.

The two vectors are identified using references to predefined variable definitions. The syntax for the **outerproduct** operator is:

```
<outerproduct/>
<ci>vector1</ci>
<ci>vector2</ci>
```

Figure 15 illustrates the procedure for computing the outer-product of two vectors.

```
<variableDef name="outerProduct"
             varID="outerProduct" units="">
  <description>
    This represents an example of computing the outer-product
    of two vectors
    vector1: [3];
    vector2: [4];
    result = matrix1: [3,4];
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of rows in the matrix -->
    <dim>4</dim> <!-- Number of columns in the matrix -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <outerproduct/>
        <ci>vector1</ci>
        <ci>vector2</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

Figure 15: An example of computing the outer-product of two vectors

4.7 Selector

The **selector** operator identifies those entries to extract from a vector, as well as identifying rows, columns, diagonals or sub-matrices to extract from a matrix. The extraction type (i.e. a single entry, a row, a column, etc.) is defined using the **other** attribute of the **selector** operator by assigning it one of the descriptors listed in Table 2. The extraction types are discussed in Sections 4.7.1 to 4.7.5 of this report.

*Table 2: Descriptors that may be used when extracting entries from a vector or matrix using the **selector** operator*

element	To extract a single entry from a vector or matrix
row	To extract a row(s) from a matrix
column	To extract a column(s) from a matrix
diag	To extract a diagonal(s) from a matrix
mslice	To extract a sub-matrix from a matrix

The syntax for the **selector** operator is:

```
<selector other="option"/>
<ci>argument_1</ci>
<cn>argument_2</cn>
<cn>...</cn>
```

The dimensions of the originating vector or matrix, together with the choice of extraction type, define the number of arguments associated with the **selector** operator. For example, the operator requires two arguments in order to extract a single element from a vector. The first argument specifies the variable index (***varID***) of the originating vector, and the second argument defines the index[4] of the entry to be extracted, as illustrated in Figure 16 (a). Similarly, the list of arguments for extracting a row or column from a two-dimensional matrix must specify both the originating matrix and the index of the row or column to be extracted. Extracting a diagonal from a two-dimensional matrix requires three arguments to be specified, namely the originating matrix, and the row and column indices of the starting entry for the diagonal. This is also the case for extracting a sub-matrix from a larger two-dimensional matrix. Multiple non-consecutive entries from a vector and/or matrix can be extracted by grouping arguments for each extraction within XML **<apply></apply>** element definitions. Examples illustrating the extraction of multiple non-consecutive entries from vectors and matrices are presented in Appendix A.

[4] The numbering of entries in a vector, as well as the rows and columns of matrices, starts at 1 and increases using a unit increment, [4].

4.7.1 Element

The **element** descriptor provides the ability to extract a single entry from a vector or matrix. One argument is specified for each dimension of the vector or matrix in order to nominate the element to be extracted. The result is a scalar. Figure 16 illustrates the procedure for extracting single entries from a vector and a two-dimensional matrix.

```
<variableDef name="vectorElement" varID="vectorElementID">
  <calculation>
    <math>
      <apply>
        <selector other="element"/>
          <ci>vectorID</ci>
          <cn>2</cn>  <!-- Entry 2 of vectorID -->
      </apply>
    </math>
  </calculation>
</variableDef>
```

(a) Extracting an entry from a vector,

```
<variableDef name="matrixElement" varID="matrixElementID">
  <calculation>
    <math>
      <apply>
        <selector other="element"/>
          <ci>matrixID</ci>
          <cn>2</cn>  <-- Row number of Entry in matrixID -->
          <cn>2</cn>  <-- Column number of Entry in matrixID -->
      </apply>
    </math>
  </calculation>
</variableDef>
```

(b) Extracting an entry from a two-dimensional matrix,

Figure 16: The syntax for extracting an entry from a vector and matrix

4.7.2 Row

The **row** descriptor provides the ability to extract a row(s), or part of a row, from a matrix. An argument is specified for each plane of the matrix if it has more than two dimensions, together with an argument indicating the nominated row to be extracted. The result is a vector when only one row is extracted; otherwise, the result is a matrix. The value specifying the variable's dimension (through **dimensionDef**), or ***column*** dimension when extracting multiple rows, indicates the number of elements to be extracted from the nominated row, starting from the first entry. Figure 17 illustrates the procedure for extracting a row from a matrix.

```
<variableDef name="matrixRow" varID="matrixRowID">
  <description>
    Extracting row 2 from the 4th plane of a three-dimensional
    matrix, [4,3,3].
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of entries in the result vector -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="row"/>
          <ci>matrixID</ci>
          <cn>4</cn>  <!-- Plane 4 of matrixID -->
          <cn>2</cn>  <!-- Row 2 of [3,3] matrix -->
      </apply>
    </math>
  </calculation>
</variableDef>
```

(a) extracting a row from a three-dimensional matrix,

```
<variableDef name="matrixRow" varID="matrixRowID">
  <dimensionDef>
    <dim>3</dim> <!-- Number of entries in the result vector -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="row"/>
          <ci>matrixID</ci>
          <cn>2</cn>  <!-- The 1st 3 entries of row 2 -->
      </apply>
    </math>
  </calculation>
</variableDef>
```

(b) extracting a partial row from a [3,4] matrix,

Figure 17: The syntax for extracting a row from a matrix

4.7.3 Column

The **column** descriptor provides the ability to extract a column(s), or part of a column, from a matrix. An argument is specified for each plane of the matrix if it has more than two dimensions, together with an argument indicating the nominated column to be extracted. The result is a vector when only one column is extracted; otherwise the result is a matrix. The value specifying the variable's dimension (through **dimensionDef**), or *row* dimension when extracting multiple columns, indicates the number of elements to be extracted from the nominated column, starting from the first entry. Figure 18 illustrates the procedure for extracting a column from a matrix.

```
<variableDef name="matrixColumn" varID="matrixColumnID">
  <description>
    Extracting column 2 from two-dimensional matrix, [4,3].
  </description>
  <dimensionDef>
    <dim>4</dim> <!-- Number of entries in the result vector -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="column"/>
          <ci>matrixID</ci>
          <cn>2</cn>  <!-- Column 2 of [4,3] matrix -->
      </apply>
    </math>
  </calculation>
</variableDef>
```

(a) extracting a column from a two-dimensional matrix,

```
<variableDef name="matrixColumn" varID="matrixColumnID">
  <dimensionDef>
    <dim>3</dim> <!-- Number of entries in the result vector -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="column"/>
          <ci>matrixID</ci>
          <cn>1</cn>  <!-- The 1st 3 entries of column 1 -->
      </apply>
    </math>
  </calculation>
</variableDef>
```

(b) extracting a partial column from a [4,3] matrix,

Figure 18: The syntax for extracting a column from a matrix

4.7.4 Diag

The **diag** descriptor provides the ability to extract a planar diagonal(s), or part there of, from a matrix. An argument is specified for each plane of the matrix if it has more than two dimensions, together with an argument indicating the starting row number and an argument indicating the starting column number. The result is a vector when only one diagonal is extracted; otherwise the result is a matrix with the diagonals arranged as the columns of the matrix. The value specifying the variable's dimension (through **dimensionDef**), or *row* dimension when extracting multiple diagonals, indicates the number of elements to be extracted from the nominated diagonal(s). Figure 19 illustrates the procedure for extracting a diagonal from a matrix.

```
<variableDef name="matrixDiagonal" varID="matrixDiagonalID">
  <dimensionDef>
    <dim>2</dim>
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="diag"/>
          <ci>matrixID</ci>
          <cn>1</cn> <!-- Row number of initial entry -->
          <cn>1</cn> <!-- Column number of initial entry -->
      </apply>
    </math>
  </calculation>
</variableDef>
```

Figure 19: The syntax for extracting a diagonal vector from a two-dimensional [2,2] matrix

4.7.5 Mslice

The **mslice** descriptor provides the ability to extract one matrix from another matrix, where the dimension of the extracted matrix is equal to or less than the dimension of the original matrix. The number of entries extracted in each dimension is specified through the variable's dimension element, **dimensionDef**. Arguments are specified by nominating where the extraction is to start, with an argument for each dimension of the original matrix. Figure 20 illustrates the procedure for extracting a three-dimension matrix from a larger matrix starting at row one and column two of the second plane.

```
<variableDef name="matrixMslice" varID="matrixMsliceID">
  <description>
    Extracting a three-dimensional [2,2,2] matrix from
    a larger matrix with dimension [4,4,3]
  </description>
  <dimensionDef>
    <dim>2</dim> <!-- Number of planes in the result matrix -->
    <dim>2</dim> <!-- Number of rows in the result matrix -->
    <dim>2</dim> <!-- Number of columns in the result matrix -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="mslice"/>
          <ci>matrixID</ci>
          <cn>2</cn>  <!-- The 2nd plane of matrixID -->
          <cn>1</cn>  <!-- Row number to start extraction -->
          <cn>2</cn>  <!-- Column number to start extraction -->
      </apply>
    </math>
  </calculation>
</variableDef>
```

Figure 20: The syntax for extracting a matrix from a larger matrix

4.8 Unsupported Vector and Matrix Operators

The MathML Version 2.0 syntax does not include operators for computing the trace, adjoint or rank of vectors or matrices. In addition, the syntax does not include operators for computing the eigenvectors and the associated eigenvalues of a matrix, or populating larger vectors and matrices using sub-vectors and matrices. The availability of these operators would assist the task of defining dynamics system models within the DAVE-ML syntax.

A method of defining elements such as **trace**, **adjoint**, **rank**, **eigenvectors**, **eigenvalues** could be introduced through a syntax that extends the MathML syntax. Alternatively, these elements could be added to the DAVE-ML syntax. However, there is a preference within the DAVE-ML community to keep the definition of mathematical operators separate from the definition of variables and data structures. Future versions of the MathML syntax may include these operators thereby alleviating the need to develop a separate syntax.

5. Input Vector and Matrix Variables

This report defines a syntax enabling the management of data as vectors and matrices within the DAVE-ML framework. Whilst not directly concerned with the syntax for defining vector and matrix variables, the following recommendations are provided as guidance for interacting with vector and matrix variables that are specified as *inputs*[5].

a. If a vector or matrix variable is defined as an input variable, then an external application should set that variable using a vector or matrix having the equivalent number of entries.

b. The DAVE-ML syntax permits scalar input variables to be set using either the **initialValue** attribute of the **variableDef**, or a scalar value from an external application, which is then stored for future reference. The ability to initialise a vector or matrix by setting the **initialValue** using a scalar value should be retained. The vector and matrix syntax supports initialising a vector or matrix with a sequence of data via the **array** sub-element. If the **array** sub-element is not specified then all entries in the vector or matrix should be set to the scalar value assigned to the **initialValue**. Similarly, if an external application sets the value of the vector or matrix using only a scalar, then all entries of the variable should be set to that value.

c. When setting a matrix using a vector it should be assumed that the vector represents a row of data for the matrix, and that all rows are set using the same data vector.

d. Setting large dimension matrices, greater than two-dimensional, using sub-matrices is not recommended.

[5] In DAVE-ML an input variable is indicated using the **isInput** sub-element when defining the **variableDef**.

6. Conclusion

DAVE-ML, together with the ANSI/AIAA-S119-2011 Flight Dynamic Model Exchange Standard, provides a framework for encoding flight vehicle simulation data packages for exchange between simulation applications and the long term storage of model data; however, it only supports scalar time-independent data.

Syntax consistent with the DAVE-ML has been developed to manage data as vectors and n-dimensional matrices. The syntax provides the capability to either directly encode data for a vector or matrix based parameter, or compute the parameter using an algebraic equation. In addition, it permits parameters having a common basis to be grouped in a simplified format, and in doing so, improve the clarity of encoding data for aerospace vehicle models.

The vector and matrix syntax will enhance the ability of the DAVE-ML to encode entire flight vehicle dynamic simulation models and their validation data, and simplify the exchange of aerospace vehicle dynamic model data between simulation applications.

7. References

[1] Jackson, E.B, "Dynamic aerospace Vehicle Exchange Markup Language, (DAVE-ML) Reference Version 2.0", AIAA Modeling and Simulation Technical Committee, *https://info.aiaa.org/tac/ASG/MSTC,* May, 2010.

[2]ANSI/AIAA-S119-2011 – Flight Dynamics Model Exchange Standard, American Institute of Aeronautics and Astronautics, Reston, VA, USA.

[3] World Wide Web Consortium (W3C) "W3C Recommendation: Extensible Markup Language (XML)", *http://www.w3.org/TR/xml/*, 2008-11-26.

[4] World Wide Web Consortium (W3C) "Mathematical Markup Language (MathML) Version 2.0 (Second Edition)", *http://www.w3.org/TR/MathML2/*, 2003.

[5] John H. Hubbard, Barbara Burke Hubbard, *Vector Calculus, Linear Algebra and Differential Forms – A Unified Approach*, Prentice Hall, New Jersey, 1999.

[6] Benjamin C. Kuo, Automatic Control Systems 5th Ed, Prentice Hall, New Jersey, 1987.

Appendix A: Examples

The examples presented in this Appendix supplement those presented previously in the body of this report. They define vectors and matrices using the DAVE-ML syntax extensions, as well as apply vector and matrix operations to compute new variables.

A.1 Example Scalar Variable Definitions

These scalar variable definitions are included here as they are used during the definition of the vector and matrix examples.

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      Scalar Variable Definitions: (used by vector and matrix definitions)
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<!-- Inputs Variables -->
<variableDef name="fuelMass_kg" varID="fuelMass" units="kg"
             initialValue="1.0">
  <description>
    Represents a consumable fuel mass for the body
  </description>
  <isInput>
</variableDef>

<variableDef name="eulerInclinationAngle_rad"
             varID="eulerInclinationAngle"
             units="rad" initialValue="0.08726">
  <description>
    Represents the Euler inclination angle - Theta
  </description>
  <isInput>
</variableDef>

<variableDef name="eulerRollAngle_rad" varID="eulerRollAngle"
             units="rad" initialValue="0.52356">
  <description>
    Represents the Euler roll (bank) angle - Phi
  </description>
  <isInput>
</variableDef>

<variableDef name="eulerAzimuthAngle_rad" varID="eulerAzimuthAngle"
             units="rad" initialValue="0.052356">
  <description>
    Represents the Euler azimuth (yaw) angle - Psi
  </description>
  <isInput>
</variableDef>
```

```
<!-- Internal Variables -->
<variableDef name="inertiaIXX_kgm2" varID="inertiaIXX" units="kgm2">
  <description>
    This represents the XX inertia of the body
  </description>
  <calculation>
    <math>
      <apply>
        <plus/>
        <apply>
          <times/>
          <cn>2000.0</cn>
          <ci>fuelMass_</ci>
        </apply>
        <cn>30000.0</cn>
      </apply>
    </math>
  </calculation>
</variableDef>

<variableDef name="inertiaIYY_kgm2" varID="inertiaIYY" units="kgm2">
  <description>
    This represents the YY inertia of the body
  </description>
  <calculation>
    <math>
      <apply>
        <plus/>
        <apply>
          <times/>
          <cn>200.0</cn>
          <ci>fuelMass_</ci>
        </apply>
        <cn>3000.0</cn>
      </apply>
    </math>
  </calculation>
</variableDef>

<variableDef name="inertiaIZZ_kgm2" varID="inertiaIZZ" units="kgm2">
  <description>
    This represents the ZZ inertia of the body
  </description>
  <calculation>
    <math>
      <apply>
        <plus/>
        <apply>
          <times/>
          <cn>400.0</cn>
          <ci>fuelMass_</ci>
        </apply>
        <cn>6000.0</cn>
      </apply>
    </math>
  </calculation>
</variableDef>
```

```
<variableDef name="inertiaIXZ_kgm2" varID="inertiaIXZ"
             units="kgm2" initialValue="540.0">
  <description>
    This represents the XZ inertia of the body
  </description>
</variableDef>

<variableDef name="inertiaIXY_kgm2" varID="inertiaIXY"
             units="kgm2" initialValue="650.0">
  <description>
    This represents the XY inertia of the body
  </description>
</variableDef>

<variableDef name="inertiaIYZ_kgm2" varID="inertiaIYZ"
             units="kgm2" initialValue="2300.0">
  <description>
    This represents the YZ inertia of the body
  </description>
</variableDef>
```

A.2 Examples of Vector Variables Definitions

The following are sample vector variable definitions. The first is a vector of numeric values, whilst the second is a vector containing references to variables that were previously defined. It should also be noted that mixing numeric and reference entries is acceptable.

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Vector Variable Definitions:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<variableDef name="vector1_nd" varID="vector1" units="nd">
  <description>
    An example of encoding a vector of data in
    a DAVE-ML XML file.
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of entries -->
  </dimensionDef>
  <array>
    <dataTable>1.0, 2.0, 3.0</dataTable>
  </array>
  <isOutput/>
</variableDef>

<variableDef name="eulerAngle_rad" varID="eulerAngles" units="rad">
  <description>
    An example of encoding a vector of references in
    a DAVE-ML XML file.
  </description>
  <dimensionRef dimID="vector_3">
  <array>
    <dataTable>
```

```
      eulerAzimuthAngle,
      eulerInclinationAngle,
      eulerRollAngle,
    </dataTable>
  </array>
  <isOutput/>
</variableDef>
```

A.3 Examples of Matrix Variable Definitions

The following are sample matrix variable definitions. The first is a two-dimensional matrix of numeric values, whilst the second is a matrix containing references to variables that were previously defined. The second matrix represents the inertia tensor of a body. It should also be noted that mixing numeric and reference entries is acceptable.

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Matrix Variable Definitions:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<!-- Internal Variable -->
<variableDef name="identityMatrix" varID="identityMatrix" units="">
  <description> An identity matrix. </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of rows -->
    <dim>3</dim> <!-- Number of columns -->
  </dimensionDef>
  <array>
    <dataTable>
      1.0, 0.0, 0.0, <!-- Row 1 -->
      0.0, 1.0, 0.0, <!-- Row 2 -->
      0.0, 0.0, 1.0, <!-- Row 3 -->
    </dataTable>
  </array>
</variableDef>

<!-- Output Variable -->
<variableDef name="inertiaTensor_kgm2" varID="inertiaTensor"
             units="kgm2">
  <description>
    An example of encoding a matrix of references in
    a DAVE-ML XML file.
  </description>
  <dimensionRef dimID="matrix_3x3">
  <array>
    <dataTable>
       inertiaIXX, -inertiaIXY, -inertiaIXZ, <!-- Row 1 IXX,-IXY,-IXZ -->
      -inertiaIXY,  inertiaIYY, -inertiaIYZ, <!-- Row 2 -IXY,IYY,-IYZ -->
      -inertiaIXZ, -inertiaIYZ,  inertiaIZZ, <!-- Row 3 -IXZ,-IYZ,IZZ -->
    </dataTable>
  </array>
  <isOutput/>
</variableDef>
```

A.4 Examples of Vector and Matrix Operations

The following examples use MathML vector and matrix operators to compute new variables. The examples include:

- computing the magnitude (or norm) of a vector,
- calculating the scalar (dot) and outer-products using the transpose MathML syntax,
- calculating the scalar product using the MathML syntax,
- using the inverse MathML syntax with the vector (cross) product to compute a vector, in this case the Euler angle rates,
- calculating the determinant of a square matrix,
- and extraction of various components from a matrix

A.4.1 Vector Magnitude (Norm)

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Norm Example:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<variableDef name="normValue" varID="normValue" units="">
  <description>
    This represents the magnitude (or norm) of a vector, that is the
    square root of the sum of squares of the individual elements.
    It can be computed as:
      result = square root( dot( V1, V1)), or
             = square root( scalarproduct( V1, V1)) using the MathML tags
    For vector1, defined previously, the result would be the square root
    of 14, which is approximately equal to 3.74166
  </description>
  <calculation>
    <math>
      <apply>
        <root/>
        <apply>
          <scalarproduct/>
          <ci>vector1</ci>
          <ci>vector1</ci>
        </apply>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

A.4.2 Transpose Operator

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Transpose Examples:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<variableDef name="transposeToScalar" varID="transposeToScalar" units="">
  <description>
    This represents an example of using the transpose directive to
    form a scalar. It is equivalent to the scalar product of two vectors.

    vector1 : [3,1]; eulerAngles: [3,1]
    ~vector1: [1,3]; eulerAngles: [3,1]
    result = ~vector1 * eulerAngles: [1,1] (scalar)
  </description>
  <calculation>
    <math>
      <apply>
        <times/>
        <apply>
          <transpose/>
          <ci>vector1</ci>
        </apply>
        <ci>eulerAngles</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>

<variableDef name="transposeToMatrix" varID="transposeToMatrix" units="">
  <description>
    This represents an example of the transpose directive to form a
    matrix. It is equivalent to the 'outerproduct' of two vectors.

    vector1: [3,1]; eulerAngles: [3,1]
    vector1: [3,1]; ~eulerAngles: [1,3]
    result = vector1 * ~eulerAngles: [3,3]
  </description>
  <dimensionRef dimID="matrix_3x3">
  <calculation>
    <math>
      <apply>
        <times/>
        <ci>vector1</ci>
        <apply>
          <transpose/>
          <ci>eulerAngles</ci>
        </apply>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

A.4.3 Inverse and Vector-Product Operators

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Inverse & Vector (Cross) product Examples:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<variableDef name="eulerRates_rad_s" varID="eulerRates" units="rad_s">
  <description>
    This represents the calculation of the Euler rates from known
    angular velocities, the inertia tensor and the applied moments.
    result = !inertiaTensor *
             (( -angularVelocity X ( inertiaTensor * angularVelocity)) +
             moment)

    MathML defines the cross product as the 'vectorproduct',
    which is the operator used in this example.
  </description>
  <dimensionDef>
      <dim>3</dim> <!-- Number of entries -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <times/>
        <apply>
          <inverse/>
          <ci>inertiaTensor</ci>
        </apply>
        <apply>
          <plus/>
          <apply>
            <vectorproduct/>
            <apply>
              <minus/>
              <ci>angularVelocity</ci>
            </apply>
            <apply>
              <times/>
              <ci>inertiaTensor</ci>
              <ci>angularVelocity</ci>
            </apply>
          </apply>
          <ci>moment</ci>
        </apply>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

A.4.4 Scalar-Product Operator

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Scalar (Dot) Product Examples:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<variableDef name="dotProduct" varID="dotProduct" units="">
  <description>
    This represents the calculation of the dot product of two vectors.
    The result is a scalar.
    MathML defines the dot product as the 'scalarproduct', which is the
    operator used in this example.
  </description>
  <calculation>
    <math>
      <apply>
        <scalarproduct/>
        <ci>vector1</ci>
        <ci>eulerAngles</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

A.4.5 Determinant Operator

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Determinant Examples:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<variableDef name="determinantInertia" varID="determinantInertia"
             units="">
  <description>
    This represents the calculation of the determinate of a matrix.
    The matrix must be square to compute the determinant. The result
    is a scalar.
  </description>
  <calculation>
    <math>
      <apply>
        <determinant/>
        <ci>inertiaTensor</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

A.4.6 Selector Operator

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Selector Examples:
     An extension to the MathML 'selector' operator has been developed. The examples
     shown here use the 'other' attribute to provide context for the 'selector' operator
     having entries of either:
     element     : for the extraction of an entry,
     row         : for the extraction of a row(s) from a matrix,
     column      : for the extraction of a column(s) from a matrix,
     diag        : for the extraction of a diagonal(s) vector from a matrix,
     mslice      : for the extraction of a sub-matrix.

     The number of arguments that need to be set depends on the selection of the 'other'
     attribute and the dimension of the vector or matrix. The argument immediately after
     the 'selector' operator is the name of the vector or matrix from which data is to be extracted.
     The following arguments define the plane, row or column to be extracted, or where
     extraction starts.
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Extracting elements:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<variableDef name="elementSelection" varID="elementSelection" units="">
  <description>
    This represents the process for extracting an entry from
    a vector or matrix. The result is a scalar.
    The entry extracted is [IYY].
    MathML numbers rows and columns of vectors and matrices from 1.
    Thus IYY is element [2,2]
  </description>
  <calculation>
    <math>
      <apply>
        <selector other="element"/>
        <ci>inertiaTensor</ci>
        <!-- Need one argument for each dimension of the matrix -->
        <cn>2</cn> <!-- Row number of entry -->
        <cn>2</cn> <!-- Column number of entry -->
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Extracting rows:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<variableDef name="rowSelection" varID="rowSelection" units="">
  <description>
    This represents the process for extracting a row from a matrix.
    The result is a vector. The row extracted is [-IXZ, -IYZ, IZZ].
    MathML numbers rows and columns of vectors and matrices from 1.
    Thus the row is #3
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of entries in the resulting vector -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="row"/>
        <ci>inertiaTensor</ci>
        <cn>3</cn> <!-- Row number to extract -->
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>


<variableDef name="multiRowSelection" varID="multiRowSelection" units="">
  <description>
    This represents the process for extracting two rows from a matrix.
  </description>
  <dimensionDef>
    <dim>2</dim> <!-- Number of rows in resulting matrix -->
    <dim>3</dim> <!-- Number of columns in resulting matrix -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="row"/>
        <apply> <!-- 1st row to extract -->
          <ci>inertiaTensor</ci>
          <cn>3</cn> <!-- Row number to extract -->
        </apply>
        <apply> <!-- 2nd row to extract -->
          <ci>inertiaTensor</ci>
          <cn>1</cn> <!-- Row number to extract -->
        </apply>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Extracting columns:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<variableDef name="columnSelection" varID="columnSelection" units="">
  <description>
    This represents the process for extracting a column from a matrix.
    The result is a vector. The column extracted is [-IXY, IYY, -IYZ].
    MathML numbers rows and columns of vectors and matrices from 1.
    Thus the column is #2
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of entries in the resulting vector -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="column"/>
        <ci>inertiaTensor</ci>
        <cn>2</cn> <!-- Column number to extract -->
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>


<variableDef name="multiColSelection" varID="multiColSelection" units="">
  <description>
    This represents the process for extracting two columns from a matrix.
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of rows in resulting matrix -->
    <dim>2</dim> <!-- Number of columns in resulting matrix -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="column"/>
        <apply> <!-- 1st column to extract -->
          <ci>inertiaTensor</ci>
          <cn>1</cn> <!-- Column number to extract -->
        </apply>
        <apply> <!-- 2nd column to extract -->
          <ci>inertiaTensor</ci>
          <cn>3</cn> <!-- Column number to extract -->
        </apply>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Extracting diagonals:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<variableDef name="diagSelection" varID="diagSelection" units="">
  <description>
    This represents the process for extracting a planar diagonal
    from a matrix. The diagonal extracted is [IXX, IYY, IZZ].
    MathML numbers rows and columns of vectors and matrices from 1.
    The starting element in the matrix is [1,1]
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- The number of entries in the resulting vector -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="diag"/>
        <ci>inertiaTensor</ci>
        <cn>1</cn> <!-- Row number to start extraction -->
        <cn>1</cn> <!-- Column number to start extraction -->
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>


<variableDef name="multiDiagSelection" varID="multiDiagSelection"
             units="">
  <description> This represents the process for extracting two
     diagonals from a matrix.
  </description>
  <dimensionDef>
    <dim>2</dim> <!-- Number of rows in resulting matrix -->
    <dim>2</dim> <!-- Number of columns in resulting matrix -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="diag"/>
        <apply>  <!-- 1st diagonal to extract -->
          <ci>inertiaTensor</ci>
          <cn>2</cn> <!-- Row number to start extraction -->
          <cn>1</cn> <!-- Column number to start extraction -->
        </apply>
        <apply> <!-- 2nd diagonal to extract -->
          <ci>inertiaTensor</ci>
          <cn>1</cn> <!-- Row number to start extraction -->
          <cn>2</cn> <!-- Column number to start extraction -->
        </apply>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Extracting a matrix:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<variableDef name="matrixSlice" varID="matrixSlice" units="">
  <description>
    This represents the process for extracting a matrix from a matrix.
    The result is a matrix.
    The dimension of the extracted matrix is equal to or less than the
    dimension of the original matrix. The number of entries extracted
    in each dimension is defined by the dimensionDef properties for
    the sub-matrix.

    The element from which to start extracting the sub-matrix needs
    to be defined.

    The matrix extracted is [IXX, -IXY; -IXY, IYY].
    MathML numbers rows and columns of vectors and matrices from 1.
    The starting element in the matrix is [1,1],
  </description>
  <dimensionDef>
    <dim>2</dim> <!-- The number of rows in sub-matrix -->
    <dim>2</dim> <!-- The number of columns in sub-matrix -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="mslice"/>
        <ci>inertiaTensor</ci>
        <!-- Need one argument for the starting row of the matrix -->
        <!-- Need one argument for the starting column of the matrix -->
        <cn>1</cn> <!-- Row number to start extraction -->
        <cn>1</cn> <!-- Column number to start extraction -->
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

Vector and Matrix Variable Definitions in DAVE-ML

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

DSTO Defence Science and Technology Organisation
506 Lorimer St
Fishermans Bend Victoria 3207 Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-TN-1146	AR-015-431	Technical Note	December 2012

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
2011/1038268/1	AVD-MSTS4	Chief, Air Vehicles Division	37	6

13. DSTO Publications Repository

http://dspace.dsto.defence.gov.au/dspace/

14. RELEASE AUTHORITY

Chief, Air Vehicles Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS Yes

18. DSTO RESEARCH LIBRARY THESAURUS http://web-vic.dsto.defence.gov.au/workareas/library/resources/dsto_thesaurus.shtml

Aerospace simulation, Modelling standards, Aircraft flight dynamics data exchange

19. ABSTRACT

The Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML) is a syntactical language for exchanging flight vehicle dynamic model data. It has been developed in conjunction with the ANSI/AIAA S-119-2011 Flight Dynamics Model Exchange Standard prepared by the American Institute of Astronautics and Aeronautics (AIAA) Modeling and Simulation Technical Committee (MSTC). The purpose of DAVE-ML is to provide a framework to encode entire flight vehicle simulation data packages for exchange between simulation applications and the long-term archiving of model data. This report documents an extension to DAVE-ML for managing data as vectors or n-dimensional matrices. The attributes and dependencies of the additional DAVE-ML elements are discussed, and examples of encapsulating data as vectors and matrices are provided.