Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML) Reference

Version 2.0.1

2011-03-31

AIAA Modeling and Simulation Technical Committee [https://info.aiaa.org/tac/ASG/MSTC]

E. Bruce Jackson, NASA Langley Research Center <bruce.jackson@nasa.gov>

Abstract

The Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML) is a text-based file format intended for encoding the principal elements of a flight simulation model for an aerospace vehicle. It is based on two other open standards: the Extensible Markup Language (XML) version 1.1 and the Mathematical Markup Language (MathML) version 2.0, both products of the World Wide Web Consortium. DAVE-ML defines additional grammar (markup elements) to provide a domain-specific language capable of aerospace flight dynamics modeling, verification, and documentation.

This markup language represents the encoding format for ANSI/AIAA S-119-2011 Flight Dynamic Model Exchange Standard [AIAA11].

This is first release version of the reference manual for DAVE-ML syntax and markup. DAVE-ML syntax is specified by the `DAVEfunc.dtd` Document Type Definition (DTD) file; the version number above refers to the version of the `DAVEfunc.dtd`.

DAVE-ML is an open standard developed by the American Institute of Aeronautics and Astronautics (AIAA) Modeling and Simulation Technical Committee (MSTC). Contact the author above for more information or comments regarding refinement of DAVE-ML.

Table of Contents

1. Changes to this document

This section contains a list of the changes since version 2.0 Release Candidate 4 of the DAVE-ML DTD.

1.1. Changes since version 2.0RC4

This section outlines changes to the DTD and this reference manual since version 2, release candidate 4, was released in January 2011.

The changes are divided into *structural* changes to the DTD and *non-structural* changes to the DTD and reference manual.

1.1.1. DTD structural changes since version 2.0

- None. Changed comment in header to reflect end of formal development.

1.1.2. Non-structural DTD and reference manual changes since version 2.0

- Changed "BSR/AIAA" designation of the S-119 standard to "ANSI/AIAA" to reflect released document designation.

1.1.3. DTD structural changes since version 2.0RC4

- None.

1.1.4. Non-structural DTD and reference manual changes since version 2.0RC4

- Corrected the function definition element overview BNF to reflect that `utID`, `gtID` attributes are mandatory in table definitions (changed went into effect with release candidate 4, but this change was not in the original RC4 reference manual).

- Removed all of the change history except for the most recent version, as it was getting rather longish and inappropriate for this initial release.

- In several places, changed verbiage to indicate S-119 is no longer a draft document.

2. Introduction

This document describes the format for DAVE-ML model definition files. DAVE-ML is a proposed standard format for the interchange of aerospace vehicle flight dynamics models. The intent of DAVE-ML is to significantly expedite the process of re-hosting a simulation model from one facility to another and function as an improved method to promulgate changes to a particular model between various facilities.

DAVE-ML is based on the Extensible Markup Language (XML), a World-Wide Web Consortium (W3C) standard. More information on XML is available here [http://www.w3.org/XML/].

The exchange of aerospace vehicle flight dynamics models may derive many benefits from the application of XML in general, and DAVE-ML in particular:

- Provides a human-readable text description of the model
- Provides an unambiguous machine-readable model description, suitable for conversion into programming language or direct import into object-oriented data structures at run-time
- Allows use of the same source file for computer-aided design and real-time piloted simulation
- Based on open, non-proprietary, standards that are language- and facility-independent
- Allows inclusion of statistical properties, such as confidence bounds and uncertainty ranges, suitable for Monte Carlo or other statistical analysis of the model
- Complies with emerging AIAA simulation data standards
- Represents a self-contained, complete, archivable data package, including references to reports, wind-tunnel tests, author contact information, and data provenance
- Is self-documenting and easily convertible to on-line and hard-copy documentation

A more complete discussion on the benefits and design of DAVE-ML can be found at the DAVE-ML web site: *http://daveml.org* [http://daveml.org]

3. Purpose

DAVE-ML is intended to encode (for exchange and long-term archive) an entire flight vehicle dynamic simulation data package, as is traditionally done in initial delivery and updates to engineering development, flight training, and accident investigation simulations. It is intended to provide a programming-language-independent representation of the aerodynamic, mass/inertia, landing gear, propulsion, and guidance, navigation and control laws for a particular vehicle.

Traditionally, flight simulation data packages are often a combination of paper documents and data files on magnetic or optical media. This collection of information is very much vendor-specific and is often incomplete or inconsistent. Many times, the preparing facility makes incorrect assumptions about how the receiving facility's simulation environment is structured. As a result, the re-hosting of the dynamic flight model by the receiving facility can take weeks or longer as the receiving facility staff learns the contents and arrangement of the data package, the model structure, the various data formats, and variable names/units/sign conventions. The staff then spends additional time running check-cases (if any were included in the transmittal) and tracking down inevitable differences in results.

There are obvious benefits to automating most of this tedious, manual process. Often, when a pair of facilities has already exchanged one model, the transmission of another model is much faster since the receiving facility will probably have devised some scripts and processes to convert the data (both model and check-case data).

The purpose of DAVE-ML is to develop a common exchange format for these flight dynamic models. The advantage gained is to enable any simulation facility or laboratory, after having written a DAVE-ML import and/or export script, to automatically receive and/or transmit such packages (and updates to those packages) rapidly with other DAVE-ML-compliant facilities.

To accomplish this goal, the DAVE-ML project is starting with the bulkiest part of most aircraft simulation packages: the aerodynamics model. This initial version of DAVE-ML can be used to transport a complete aerodynamics model, including descriptions of the aerodynamic build-up equations and data tables, and include references to the documentation about the aerodynamics model and check-case data. This format also lends itself to any static subsystem model (i.e. one that contains no state vector) such as the mass and inertia model, or a weapons load-out model, or perhaps a navigational database. The only requirement is that model outputs must be unambiguously defined in terms of inputs, with no past history (state) information required.

DAVE-ML forms the encoding portion of the Flight Dynamic Model Exchange Standard, ANSI/AIAA S-119-2011. More information is available at the S-119 web site [AIAA11].

4. Background

The idea of a universally understood flight dynamics data package has been discussed for at least two decades within the AIAA technical committees [Hildreth94], [Hildreth98]. There have been proposals in the past to standardize on Fortran as well as proprietary, vendor-specified modeling packages, including graphical ones. The National Aerospace Plane (NASP) Program, under the guidance of Larry Schilling of NASA Dryden Flight Research Center, developed a hybrid Web- and secure-FTP-based system for exchanging NASP subsystem models as well as a naming convention for variables, file names, and other simulation components in the early 1990s. Some other simulation standards have subsequently been proposed by the AIAA and are under active consideration at this writing [AIAA11].

4.1. Existing standards

The AIAA has published a Recommended Practice concerning sign conventions, axes systems, and symbolic notation for flight-vehicle models [AIAA92].

The AIAA Modeling and Simulation Technical Committee has developed a standard for the exchange of simulation modeling data. This includes a methodology for accomplishing the gradual standardization of simulation model components, a mechanism for standardizing variable names within math models, as well as a convention for encoded units-of-measure. This document is included as an Annex to the standard [AIAA01], [AIAA03], [AIAA11].

4.2. DAVE-ML proposal

In a 2002 AIAA paper, Jackson and Hildreth proposed using XML to exchange flight dynamic models [Jackson02]. This paper gave outlines for how such a standard could be accomplished, and provided a business justification for pursuing such a goal.

The 2002 proposal included several key aspects from the draft standard, including allowing use of a standard variable-name convention and data table schema and including traceability for each data point back to a referenced document or change order.

In a subsequent paper, Jackson, Hildreth, York and Cleveland [Jackson04] reported on the results of a demonstration using DAVE-ML to exchange two aerodynamic models between simulation facilities, showing the feasibility of the idea.

4.3. Recent applications

Several successful applications of DAVE-ML have been reported. These include the adoption of DAVE-ML by the Australian DSTO for threat models [Brian05] and the U.S. Navy for their Next Generation Threat System [Hildreth08]. Import tools to allow the direct use of DAVE-ML models (without recompilation) in real-time piloted simulations have been reported by NASA Langley Research Center (LaRC) [Hill07] and at NASA Ames Research Center (ARC). Some interest has been generated within NASA's Orion Project as well [Acevedo07]. Other applications include TSONT, a trajectory optimization tool ([Durak06]) and aircraft engine simulations ([Lin04]).

DAVE-ML format for models has also been supported by the GeneSim [http://genesim.sourceforge.net] Project, which is providing open-source utility programs that realize a DAVE-ML model in object-oriented source code such as C++, Java and C#.

5. Supporting technologies

DAVE-ML relies on MathML, version 2.0, to define mathematical relationships. MathML-2 is an XML grammar for describing mathematics as a basis for machine-to-machine communication. It is used in DAVE-ML to describe relationships between variables and function tables and may also be used for providing high-quality typeset documentation from the DAVE-ML source files. More information is available at the MathML-2 home web page, found at http://www.w3.org/Math/.

MathML-2 provides a mostly complete set of mathematical functions, including trigonometric, exponential and switching functions. One function that is available in most programming languages and computer-aided design tools but is missing from MathML-2 is the two-argument arctangent function which provides a continuous angle calculation by comparing the sine and cosine components of a 2D coordinate set. DAVE-ML provides a means of extending MathML-2 for a predefined set of functions (currently only the *atan2* function is defined). Thus, a DAVE-ML-compliant processing tool should recognize this extension (which is accomplished by using the MathML-2 `csymbol` element). See Section 6.2.2 (p. 15) for a discussion and Example 6 (p. 22).

6. Major elements

At present, only one major element of DAVE-ML has been defined: the function definition element, or `DAVEfunc`. `DAVEfunc` is used to describe static models such as aerodynamic and inertia/mass models, where an internal state is not included. Static check-cases can also be provided for verification of proper implementation.

Other major elements are envisioned to describe dynamic portions of the vehicle model (such as propulsion, landing gear, control systems, etc.) and dynamic check-case (time history) data. Ultimately DAVE-ML should be capable of describing a complete flight-dynamics model with sufficient data to validate the proper implementation thereof.

6.1. The `DAVEfunc` major element

The `DAVEfunc` element contains both data tables and equations for a particular static model. A `DAVEfunc` element is broken into six components: a file header, variable definitions, breakpoint definitions, table definitions, function definitions and optional check-cases. This decomposition reflects common practice in engineering development flight-simulation models in which the aerodynamic database is usually captured in multi-dimensional, linearly interpolated function tables. The inputs to these tables are usually state variables of the simulation (such as Mach number or angle-of-attack). The outputs from these interpolated tables are combined to represent forces and moments acting on the vehicle due to aerodynamics.

It is possible, using `DAVEfunc` and MathML-2 elements, to completely define an aerodynamic model without use of function tables (by mathematical combinations of input variables, such as a polynomial model) but this is not yet common in the American flight-simulation industry.

A `fileHeader` element is included to give background and reference data for the represented model.

Variables, or more properly *signals*, are used to route inputs and calculations through the subsystem model into outputs. Each variable is defined with a `variableDef` element. Variables can be thought of as parameters in a computer program or signal paths on a block diagram. They can be inputs to the subsystem model, constant values, outputs from the model, and/or the results of intermediate calculations. Variables must be defined for each input and output of any function element as well as any input or output of the subsystem represented. MathML-2 [http://www.w3.org/Math] *content* markup can be used to define constant, intermediate, or output variables as mathematical combination of constant values, function table outputs, and other variables, but any *presentation* markup is not required and should be ignored by the processing application (except as required to generate documentation). Variables also represent the current value of a function (the `dependentVariableDef` in a `function` definition) so the output of functions can be used as inputs to other variables or functions.

Breakpoint definitions, captured in `breakpointDef` elements, consist of a list of monotonically increasing floating-point values separated by commas or white space. These sets are referenced by "gridded" function table definitions and may be referenced by more than one `function` definition.

Function table definitions, described by `griddedTableDef` and `ungriddedTableDef` elements, generally contain the bulk of data points in an aerodynamics model, and typically

represent a smooth hyper-surface representing the value of some aerodynamic non-dimensional coefficient as a function of one or more vehicle states (typically Mach number, angle-of-attack, control surface deflection, and/or angular body rates). These function tables can be either "gridded," meaning the function has a value at every intersection of each dimension's breakpoint, or "ungridded," meaning each data point has a specified coordinate location in n-space. The same table can be reused in several functions, such as a left- and right-aileron moment contribution.

Function definitions (described by `function` elements) connect breakpoint sets and data tables to define how an output signal (or dependent variable) should vary with one or more input signals (or independent variables). The valid ranges of input-signal magnitudes, along with extrapolation requirements for out-of-range inputs, can be defined. There is no limit to the number of independent variables, or function dimensionality, of the function.

Check-case data (described by a single `checkData` element) can be included to provide information to automatically verify the proper implementation of the model by the recipient. Multiple check-cases can (and should) be specified using multiple `staticShot` test-case definitions, as well as optional internal signal values within the model to assist in debugging an instantiation of the model by the recipient.

Figure 1 (p. 10) contains excerpts from an example model, showing five of the six major parts of a DAVE-ML file.

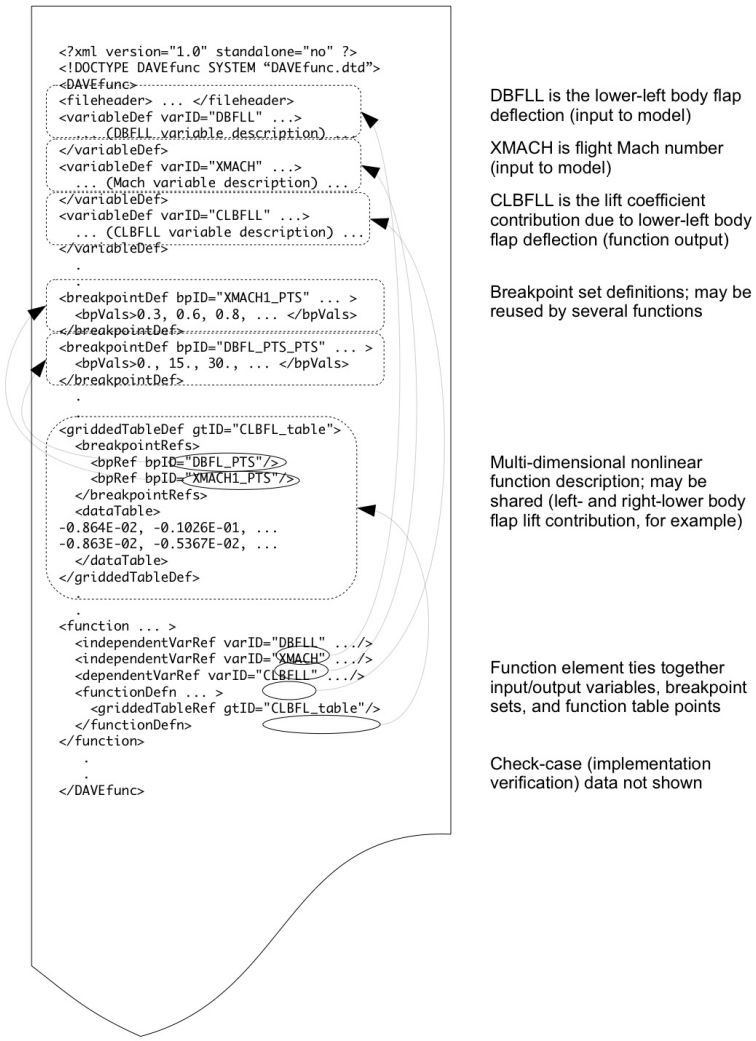

Figure 1. Excerpts from an example DAVE-ML file

A simpler version of a `function` is available in which the dependent variable breakpoint values and dependent output values are specified directly inside the `function` body. This may be preferred for models that do not reuse function or breakpoint data.

A third form of `function` is to give the gridded table values or ungridded table values inside the `function` body, but refer to externally defined breakpoint sets. This allows reuse of the breakpoint sets by other functions but keeps the table data private.

6.2. Schematic overview of `DAVEfunc`

Shown below are schematic overviews of the various elements currently available in `DAVEfunc`. Each element is described in detail in Section 8 (p. 53) later in this document. The following key is used to describe the elements and associated attributes.

```
Key:
    elementname : mandatory_attributes, [optional_attributes]
            mandatory_single_sub-element
            optional_single_sub-element?                    {editorial comment}
            ( choice_1_sub-element | choice_2_sub-element )
            zero_or_more_sub-elements*
            one_or_more_sub-elements+
            (character data) implies UNICODE text information
```

The `DAVEfunc` element has six possible sub-elements:

```
    DAVEfunc :
        fileHeader
        variableDef+
        breakpointDef*
        griddedTableDef*
        ungriddedTableDef*
        function*
        checkData?
```

`DAVEfunc` sub-elements:

`fileHeader` — This mandatory element contains information about the origin and development of this model.

`variableDef` — Each `DAVEfunc` model must contain at least one signal path (such as a constant output value). Each input, output or internal signal used by the model must be specified in a separate `variableDef`.

A signal can have only a single origin (an input block, a calculation, or a function output) but can be used (referenced) more than once as an input to one or more functions, signal calculations, and/or as a model output.

In DAVE-ML 2.0, all signals are real and scalar.

The `variableDef`s should appear in calculation order; that is, a `variableDef` should not appear before the definitions of variables upon which it is dependent. This is good practice since doing so avoids a circular reference. If a variable depends upon the output (`dependentVar`) of a `function` it can be assumed that dependence has been met, since function definitions appear later in the `DAVEfunc` element.

`breakpointDef` — A `DAVEfunc` model can contain zero, one, or more breakpoint set definitions. These definitions can be shared among several gridded function tables. Breakpoint definitions can appear in any order.

`griddedTableDef` — A `DAVEfunc` model can contain zero, one, or more gridded nonlinear function table definitions. Each table

must be used by multiple `function` definition if desired for efficiency. Alternatively, some or all `function`s in a model can specify their tables internally with an embedded `griddedTableDef` element.

A gridded function table contains dependent values, or data points, corresponding to the value of a function at the intersection of one or more breakpoint sets (one for each dimension of the table). The independent values (coordinates or breakpoint sets) are not stored within the gridded table definition but are referenced by the parent function. This allows a function table to be supported by more than one set of breakpoint values (such as left- and right-aileron deflections).

`ungriddedTableDef`

A DAVEfunc model can contain zero, one, or more ungridded nonlinear function table definitions. Unlike a rectangularly gridded table, an ungridded table specifies data points as individual sets of independent and dependent values. Each table must be used by at least one but can be used by multiple `function` definitions if necessary for efficiency. Alternatively, `function`s can retain their tables internally with a ungriddedTable element without sharing the table values with other functions.

Ungridded table values are specified as a single (unsorted) list of independent variable (input) values and associated dependent variable (output) values. While the list is not sorted, the order of the independent variable inputs is important and must match the order given in the parent `function`. Thus, functions that share an ungridded table definition must have the same ordering of independent variables.

The method of interpolating the ungridded data is not specified.

`function`

A `function` ties together breakpoint sets (for gridded-table nonlinear functions), function values (either internally or by reference to table definitions), and the input- and output-variable signal definitions, as shown in Figure 1 (p. 10). Functions also include provenance, or background history, of the function data such as wind tunnel test or other source information.

`checkData`

This optional element contains information allowing the model to be automatically verified after implementation by the receiving party.

An example of each of these sub-elements is given below. Complete descriptions of each element in detail are found in Section 8 (p. 53).

6.2.1. The file header element

The `fileHeader` element contains information about the source of the data contained within the `DAVEfunc` major element, including the author, creation date, description, reference information, and modification history.

```
fileHeader : [name]
    author+ : name, org, [email]
        contactInfo* : [contactInfoType, contactLocation]
            {text describing contact information for author}
    creationDate : date {in ISO 8601 YYYY-MM-DD format}
    fileVersion?
        {file version identifier}
    description?
        {textual description of model}
    reference* : refID, author, title, date, [classification, accession, href]
        description? :
            {textual information about reference}
    modificationRecord* : modID, date, [refID]
        author+ : name, org, [email]
            contactInfo* : [contactInfoType, contactLocation]
                {text describing contact information for author}
        description?
            {textual description of modification}
        extraDocRef* : refID
    provenance* :
            author+ : name, org, [email]
                contactInfo* : [contactInfoType, contactLocation]
                    {text describing contact information for author}
            creationDate : date {in ISO 8601 YYYY-MM-DD format}
            documentRef* : [docID,] refID
            modificationRef* : modID
            description?
                {textual description of the background of the model}
```

`fileHeader` sub-elements:

`author`	Name, organization, optional email address and other contact information for each author.
`creationDate`	Creation date of this file. See Section 6.5.4 (p. 49)for the recommended format for encoding dates.
`fileVersion`	A string that indicates the version of the document. No convention is specified for the format, but a good practice would include an automated revision number from a version control system.
`description`	An optional but recommended text description: what does this DAVE-ML file represent?
`reference`	An optional list of one or more references with a document-unique ID (must begin with alpha character), author, title, date, and optional accession and URL of the

reference. This sub-element can include a description of the reference.

`modificationRecord` — An optional list of one or more modifications with optional reference IDs, as well as author information and descriptions for each modification record. These modifications are referred to by individual function tables and/or data points, using the AIAA modification letter convention. If more than one document is associated with the modification, multiple sub-element extraDocRefs may be used in place of the `modificationRecord`'s `refID` attribute.

`provenance` — An optional list of one or more provenance elements allows the author to describe the source and history of the data within this model. Since the model may be constructed from several sources, more than one provenance may be provided, one for each source of data. Use of a `provID` attribute in the `fileHeader` is unnecessary since this provenance applies to the entire model unless otherwise specified.

Example 1. An excerpt with an example of a `fileHeader` element

```
<!--                          =================                          --> ❶
<!-- ========================   FILE HEADER   ======================== -->
<!--                          =================                          -->


  <fileHeader> ❷
    <author name="Bruce Jackson" org="NASA Langley Research Center">
      <contactInfo contactInfoType='address' contactLocation='professional'>
         MS 308 NASA, Hampton, VA 23681
      </contactInfo>
      <contactInfo contactInfoType='email' contactLocation='professional'>
        Bruce.Jackson@nospam.nasa.gov
      </contactInfo>
    </author>
    <creationDate date="2003-03-18"/> ❸

    <fileVersion>$Revision: 1.24 $</fileVersion> ❹

    <description>
     Version 2.0 aero model for HL-20 lifting body, as described in
     NASA TM-107580. This aero model was used for HL-20 approach and
     landing studies at NASA Langley Research Center during 1989-1995
     and for follow-on studies at NASA Johnson Space Center in 1994
     and NASA Ames Research Center in 2001. This DAVE-ML version was
     created in March 2003 by Bruce Jackson to demonstrate DAVE-ML.
    </description>

    <reference refID="REF01" ❺
        author="Jackson, E. Bruce; Cruz, Christopher I. & and Ragsdale, W. A."
        title="Real-Time Simulation Model of the HL-20 Lifting Body"
        accession="NASA TM-107580"
        date="1992-07-01"
```

```
    />

    <reference refID="REF02"
        author="Cleveland, William B. <nospam@mail.arc.nasa.gov>"
        title="Possible Typo in HL20_aero.xml"
        accession="email"
        date="2003-08-19"
    />

    <modificationRecord modID="A" refID="REF02"> ❻
      <author name="Bruce Jackson" org="NASA Langley Research Center">
        <contactInfo contactInfoType='address' contactLocation='professional'>
           MS 308 NASA, Hampton, VA 23681
        </contactInfo>
      </author>
      <description>
        Revision 1.24: Fixed typo in CLRUD0 function description which
        gave dependent signal name as "CLRUD1." Bill Cleveland of NASA
        Ames caught this in his xml2ftp script. Also made use of 1.5b2
        fileHeader fields and changed date formats to comply with
        convention.
      </description>
    </modificationRecord>

  </fileHeader>
```

❶ Use of comments make models more readable by humans.

❷ Start of the `fileHeader` element.

❸ Creation date of file, in ISO-8601 format. See Section 6.5.4 (p. 49)

❹ In this example, the revision number is automatically inserted by a version control system.

❺ All documents referenced by notation throughout the file should be described in the `fileHeader` as `reference` elements.

❻ All modifications made to the contents of this file should be listed in the `fileHeader` as `modificationRecord` sub-elements for easy reference by later `modificationRef` elements.

6.2.2. The variable definition element

The `variableDef` element is used to define each constant, parameter, or variable used within or generated by the defined subsystem model. It contains attributes including the variable name (used for documentation), an internal and unique `varID` identifier (used for linking inputs, functions and outputs), the units of measure of the variable, and optional axis system, sign convention, alias, and symbol declarations. Optional sub-elements include a written text description and a mathematical description, in MathML-2 content markup, of the calculations needed to derive the variable from other variables or function table outputs. Optional sub-element `isOutput` serves to indicate an intermediate calculation that should be brought out to the rest of the simulation. Another optional sub-element, `isStdAIAA`, indicates the variable name is defined in the AIAA simulation standards document. Another optional sub-element, `uncertainty`, captures the statistical properties of a (normally constant) parameter.

Optional `minValue` and `maxValue` attributes specify restrictions on the range of values the variable may have; these apply to all means of setting the value of the variable whether as an input to the model, an initial value attribute, or the result of a `calculation` sub-element. Note that the value of a variable used by a subsequent `independentVarRef` of a `function` may be further limited for use by that specific function.

Other optional sub-elements are provided to identify inputs, disturbances, and simulation control parameters, as well as the ability to identify a variable as a state or state derivative for linear model purposes.

There must be a single `variableDef` for each and every input, output or intermediate constant or variable within the DAVEfunc model.

```
variableDef : name, varID, units, [axisSystem, sign, alias, symbol,
                    initialValue, minValue, maxValue]
    description? :
            {description character data}
    (
        provenanceRef : provID
    OR
        provenance : [provID]
            author+ : name, org, [email]
                contactInfo* : [contactInfoType, contactLocation]
                    {text describing contact information}
            creationDate : date {in YYYY-MM-DD format}
            documentRef* : [docID,] refID
            modificationRef* : modID
            description?
    )?
    calculation? :
        math {defined in MathML-2 DTD}
    (isInput | isControl | isDisturbance)?
    isState?
    isStateDeriv?
    isOutput?
    isStdAIAA?
    uncertainty? : effect
        (normalPDF : numSigmas) | (uniformPDF : bounds+)
```

`variableDef` attributes:

`name`	A UNICODE name for the variable (may be the same string as the `varID`).
`varID`	An internal identifier that is unique within the file.
`units`	The units-of-measure for the signal, using the AIAA standard units convention [AIAA11].
`axisSystem`	An optional indicator of the axis system (body, inertial, etc.) in which the signal is measured. See [AIAA11] or Section 6.5 (p. 48)below for recommended practice for nomenclature.
`sign`	An optional indicator of which direction is considered positive (+RWD, +UP, etc.). See [AIAA11] or the section on Section 6.5 (p. 48) below for recommended practice for abbreviations.
`alias`	An optional, facility-specific variable name, perhaps used in the equations of motion or control system model, that

	does not conform to the AIAA standard for variable names. Use of this attribute is discouraged for portability reasons.
`symbol`	A UNICODE Greek symbol for the signal [to be superseded with more formal MathML or TeX element in a later release].
`initialValue`	An optional initial value for the parameter. This is normally specified for constant parameters only.
`minValue`	An optional minimum value that the variable may take on, regardless of how that value is determined. If present, this attribute implies a one-sided minimum value limiter that is applied to the final value of the variable. If the minimum value is greater than a sibling `maxValue` attribute the result is undetermined. This attribute must be a string representing a constant numeric value.
`maxValue`	An optional maximum value that the variable may take on, regardless of how that value is determined. If present, this attribute implies a one-sided maximum value limiter that is applied to the final value of the variable. If the maximum value is lesser than a sibling `minValue` attribute the result is undetermined. This attribute must be a string representing a constant numeric value.

`variableDef` sub-elements:

`description`	An optional text description of the variable.
`provenance`	The optional provenance element allows the author to describe the source and history of the data within this `variableDef`. Alternatively, a `provenanceRef` reference can be made to a previously defined `provenance`.
`calculation`	An optional container for the MathML-2 content markup that describes how this variable is calculated from other variables or function table outputs. This element contains a single `math` element which is defined in the MathML-2 markup language [http://www.w3.org/Math]. A MathML-2 calculation can include both constants (using the content numeric `cn` element) and references to other variables internal to the parent `DAVEfunc` description. The variables (which can include the output, or dependent variable of a `function` table) are identified using its `varID` attribute string in the appropriate MathML content identifier (`ci`) element of the expression.

	Examples of MathML expressions appear later in this document.
`isInput`	This optional element, if present, signifies that this variable is an input to the model, such as a pilot inceptor deflection or Mach number. Useful for linear model extraction tools. It must not be the result of a calculation or be cited as the dependent variable of a function.
`isControl`	This optional element, if present, signifies that this variable is a simulation control parameter, such as a trim flag or simulation time step measurement. Simulation control parameters should have no influence on the dynamic behavior of the model and should be ignored by a linear model extraction tool.
`isDisturbance`	This optional element, if present, signifies that this variable represents an external disturbance input to the model; this is useful for linear model extraction tools to partition this input separately from the other model inputs.
`isOutput`	This optional element, if present, signifies that this variable needs to be passed as an output. How this is accomplished is up to the implementer. Unless specified by this element, a variable is considered an output only if it is the result of a calculation or function AND is not used elsewhere in the `DAVEfunc`.
`isStdAIAA`	This optional element, if present, signifies that this variable is one of the standard AIAA simulation variable names that are defined as Annex A to [AIAA11]. Such identification should make it easier for the importing process to connect this variable (probably an input or output of the model) to the appropriate variable to/from the user's simulation framework.
`isState`	This optional element, if present, signifies that this variable serves as a state of the model.
`isStateDeriv`	This optional element, if present, signifies that this variable serves as a state derivative of the model.
`uncertainty`	This optional element, if present, describes the uncertainty of this parameter. See the section on Statistics below for more information about this element.

Example 2. An example of two `variableDef` elements defining input signals

In this example, two input variables are defined: `XMACH` and `DBFLL`. These two variables are inputs to a table lookup function shown in Example 11 (p. 34) below.

```
<!--                      ==========================                      -->
<!-- ====================   VARIABLE DEFINITIONS   ==================== -->
<!--                      ==========================                      -->


      <!-- ================== -->
      <!--  Input variables   -->
      <!-- ================== -->

  <variableDef name="mach"❶ varID="XMACH"❷ units="nd" symbol="M">
    <description>
        Mach number (dimensionless)
    </description>
    <isInput/>❸
    <isStdAIAA/>❹
  </variableDef>
    .
    .
    .
  <variableDef name="dbfll" varID="DBFLL" units="deg"❺ sign="TED"❻
               symbol="&#x3B4;bfll"❼>
    <description>
        Lower left body flap deflection, deg, positive trailing-edge-down (so
 deflections are
        always zero or positive).
    </description>
    <isInput/>
  </variableDef>
```

❶ The `name` attribute is intended for humans to read, perhaps as the signal name in a wiring diagram. Note that "machNumber" is one of the standard AIAA simulation variable names.

❷ The `varID` attribute is intended for the processing application to read. This is an internal identifier that must be unique within this model.

❸ The optional `isInput` attribute indicates this variable should be treated as an input to the model for model hierarchy and linear model extraction (for example).

❹ The optional `isStdAIAA` sub-element indicates this signal is one of the predefined standard variables that most simulation facilities define in their equations of motion code. The `name` attribute should correspond to the standard AIAA parameter name from Annex A of [AIAA11] or subsequent standards document

❺ The mandatory `units` attribute describes the units of measure of the variable. See Section 6.5.6 (p. 49) below for a recommended list of units-of-measure abbreviations.

❻ The optional `sign` attribute describes the sign convention that applies to this variable. In this case, the lower-left body-flap is positive with trailing-edge-down deflection. See Section 6.5.5 (p. 49) below for a recommended list of sign abbreviations.

❼ The optional `symbol` attribute allows a UNICODE character string that might be used for this variable in a symbols listing.

Example 3. A simple local variable definition example

This DAVE-ML excerpt defines `CLBFLLO` which is the dependent variable (output) from a table lookup function, shown later in Example 11 (p. 34) It is subsequently used in the calculation of the lower-left body flap lift coefficient, shown in Example 4 (p. 20).

```
      <!-- ================== -->
```

```
      <!--  Local variables   -->
      <!-- ================== -->

<!-- PRELIMINARY BUILDUP EQUATIONS -->

<!--  LOWER LEFT BODY FLAP CONTRIBUTIONS -->

<!--    table output signal   -->
  <variableDef name="CLdbfll_0" varID="CLBFLL0" units="nd">
    <description>
        Output of CLBFLL0 function; lift force contribution of
        lower left body flap deflection due to alpha^0 (constant
        term).
    </description>
  </variableDef>
```

Since this signal is not flagged as an input, control, disturbance or output, this variable is an intermediate signal local to this model.

Example 4. A more complete `variableDef` example with a `calculation` element

In this example, the local variable `CLBFLL` is defined as a calculated quantity, based on several other input or local variables including the CLBFLL0 function output variable defined in the previous example (p. 19) Note the `description` element is used to describe the equation in Fortran-ish human-readable text. The `calculation` element describes this same equation in MathML-2 content markup syntax; this portion should be used by parsing applications to create either source code, documentation, or run-time calculation structures.

```
<!--    lower left body flap lift buildup -->
  <variableDef name="CLdbfll" varID="CLBFLL" units="nd">
    <description>
        Lift contribution of lower left body flap deflection
        CLdbfll = CLdbfll_0 + alpha*(CLdbfll_1 + alpha*(CLdbfll_2
                            + alpha*CLdbfll_3)) ❶
    </description>
    <calculation>  ❷
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>  ❸
        <plus/>
          <ci>CLBFLL0</ci>  ❹
          <apply>
            <times/>
            <ci>ALP</ci>
            <apply>
              <plus/>
              <ci>CLBFLL1</ci>
              <apply>
                <times/>
                <ci>ALP</ci>
                <apply>
                  <plus/>
                  <ci>CLBFLL2</ci>
                  <apply> ❺
                    <times/>
                    <ci>ALP</ci>
                    <ci>CLBFLL3</ci>
                  </apply> <!--            a*c3   -->  ❻
                </apply> <!--        (c2 + a*c3)  -->
              </apply> <!--        a*(c2 + a*c3)  -->
            </apply> <!--    (c1 + a*(c2 + a*c3)) -->
```

```
        </apply> <!--    a*(c1 + a*(c2 + a*c3)) -->
      </apply> <!-- c0 + a*(c1 + a*(c2 + a*c3)) -->
    </math>
  </calculation>
</variableDef>
```

❶ This Fortran-ish equation, located in the `description` element, is provided in this example for the benefit of human readers; it should not parsed by the processing application.

❷ A `calculation` element always embeds a MathML-2 `math` element; note the definition of the MathML-2 namespace.

❸ Each `apply` tag pair surrounds a math operation (in this example, a `plus` operator) and the arguments to that operation (in this case, a variable `CLBFLL` defined elsewhere is added to the results of the nested `apply` operation).

❹ The content identifier (`ci`) MathML-2 element gives the `varID` of the previously defined variables used in this equation; this variable represents the output of the `CLBFLL0` function found in Example 11 (p. 34) that is captured in the `CLBFLL0` variable defined in Example 3 (p. 19) The other `ci` elements are not defined in this manual but are defined in the full model.

❺ Inner-most `apply` multiplies variables `ALP` and `CLBFLL3`.

❻ The comments here are useful for humans to understand how the equation is being built up; the processing application ignores all comments.

Example 5. Another example of an output variable based on a `calculation` element

This excerpt is an example of how an output variable (`CL`) might be defined from previously calculated local variables (in this case, `CL0`, `CLBFL`, etc.).

```
    <!-- ================== -->
    <!--  Output variables  -->
    <!-- ================== -->

<variableDef name="CL" varID="CL" units="nd" sign="+UP" symbol="CL">
  <description>
      Coefficient of lift
      CL = CL0 + CLBFUL + CLBFUR + CLBFLL + CLBFLR +
                 CLWFL + CLWFR + CLRUD + CLGE + CLLG
  </description>
  <calculation>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply> ❶
        <plus/>
        <ci>CL0</ci>
        <ci>CLBFUL</ci>
        <ci>CLBFUR</ci>
        <ci>CLBFLL</ci> ❷
        <ci>CLBFLR</ci>
        <ci>CLWFL</ci>
        <ci>CLWFR</ci>
        <ci>CLRUD</ci>
        <ci>CLGE</ci>
        <ci>CLLG</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/> ❸
</variableDef>
```

❶ Here `apply` simply sums the value of these variables, referenced by their `varID`s.

❷ This `ci` element refers to the lower left body flag lift contribution calculated in the previous example (p. 20).

❸ The `isOutput` element signifies to the processing application that this variable should be made visible to models external to this `DAVEfunc`.

Example 6. An intermediate variable with a `calculation` element that uses a DAVE-ML function extension to the default MathML-2 function set

In this excerpt, we demonstrate a means to encode a math function, *atan2*, that is not available in the default MathML-2 function set. The *atan2* function is used often in C, C++, Java and other modeling languages and has been added to the DAVE-ML standard by use of the MathML-2 `csymbol` element, specifically provided to allow extension of MathML-2 for cases such as this.

```
    <!-- ================== -->
    <!--    ATAN2 example   --> ❶
    <!-- ================== -->

<variableDef name="Wind vector roll angle" varID="PHI" units="rad">
  <description>
    This encodes the equation PHI = atan2( tan(BETA), sin(ALPHA) ) where atan2
    is the two-argument arc tangent function from the ANSI C standard math
    library.
  </description>
  <calculation>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <csymbol definitionURL="http://daveml.org/function_spaces.html#atan2"
                 encoding="text">  ❷
          atan2
        </csymbol>
        <apply>
          <tan/>
          <ci>BETA</ci>   ❸
        </apply>
        <apply>
          <sin/>
          <ci>ALPHA</ci>  ❹
        </apply>
      </apply>
    </math>
  </calculation>

</variableDef>
```

❶ This excerpt shows how to calculate wind roll angle, phi, from angle-of-attack and angle-of-sideslip; it comes from the Apollo aerodynamics data book [NAA64].

❷ The `csymbol` element is provided by MathML-2 as a means to extend the function set of MathML-2. An extension for *atan2* is the only function defined at present but others may be added to the set in the future. Note the specific URI that uniquely identifies this function; it is also the URL (web address) of the documentation of the interpretation of the *atan2* function.

❸ BETA is the `varID` of a previously defined variable.

❹ ALPHA is the `varID` of a previously defined variable.

6.2.3. The breakpoint set definition element

The breakpoint set definition element, `breakpointDef`, is used to define a list of comma- or white space-separated values that define the coordinate values along one axis of a gridded linear function value table. It contains a mandatory `bpID` attribute, an optional `name` and units-of-measure attributes, an optional text `description` element, and the comma- or white space-separated list of floating-point values in the `bpVals` element. This list must be monotonically increasing in value.

```
breakpointDef : bpID, [name, units]
    description? :
    bpVals :
        {character data of comma- or white space-separated breakpoints}
```

`breakpointDef` attributes:

`bpID`	An internal reference that is unique within the file.
`name`	A UNICODE name for the set (may be the same string as `bpID`).
`units`	The units-of-measure for the breakpoint values. See Section 6.5.6 (p. 49) below.

`breakpointDef` sub-elements:

`description`	An optional text description of the breakpoint set.
`bpVals`	A comma- or white space-separated, monotonically increasing list of floating-point values.

Example 7. Two `breakpointDef` examples in a DAVE-ML model excerpt

As an example, two breakpoint sets are defined which are used in the `function` element given below (Example 11 (p. 34**)**. Breakpoint sets `XMACH1_PTS` and `DBFL_PTS` contain values for Mach and lower body flap deflection, respectively, which are used to look up function values in several gridded function tables. One example is given below in Example 8 (p. 25).

```
<!--                          ====================                          -->
<!-- =========================   BREAKPOINT SETS   ========================= -->
<!--                          ====================                          -->


  <breakpointDef name="mach" bpID="XMACH1_PTS" units="nd">  ❶
    <description>
      Mach number breakpoints for all aero data tables
    </description>
    <bpVals>
        0.3, 0.6, 0.8, 0.9, 0.95, 1.1, 1.2, 1.6, 2.0, 2.5, 3.0, 3.5, 4.0   ❷
    </bpVals>
  </breakpointDef>
```

```
<breakpointDef name="Lower body flap" bpID="DBFL_PTS" units="deg">    ❸
  <description>Lower body flap deflections breakpoints for tables</description>
  <bpVals>0., 15., 30., 45., 60.</bpVals>
</breakpointDef>
```

❶ This `breakpointDef` element describes a Mach breakpoint set uniquely identified as `XMACH1_PTS` with no associated units of measure.
❷ The breakpoint values are given as a comma- or white space-separated list and must be in monotonically increasing numerical order.
❸ This breakpoint set defines the breakpoints for lower body flap deflection.

6.2.4. The gridded table definition element

The `griddedTableDef` element defines a multi-dimensional table of values corresponding with the value of an arbitrary function at each intersection of a set of specified independent input values. The coordinates along each dimension are defined in separate `breakpointDef` elements that are referenced within this element by `bpRef`s, one for each dimension.

The data contained within the data table definition are a comma- or white space-separated set of floating-point values. This list of values represents a multi-dimensional array whose size is inferred from the length of each breakpoint vector. For example, a 2D table that is a function of an eight-element Mach breakpoint set and a ten-element angle-of-attack breakpoint set is expected to contain 80 (8 x 10) comma- or white space-separated values.

By convention, the `breakpointRefs` are listed in order such that the last breakpoint set varies most rapidly in the associated data table listing. See Section 6.5.1 (p. 48) below.

An optional `uncertainty` element may be provided that represents the statistical variation in the values presented. See Section 6.4 (p. 42) for more information about this element.

```
griddedTableDef : gtID, [name, units]
    description?
      {description of table in character data}
    EITHER
        provenanceRef? : provID
    OR
        provenance? : [provID]
            author+ : name, org, [email]
                contactInfo* : [contactInfoType, contactLocation]
                    {text describing contact information}
            creationDate : date {in YYYY-MM-DD format}
            documentRef* : [docID,] refID
            modificationRef* : modID
            description?
    breakpointRefs :
        bpRef+ : bpID
    uncertainty? : effect
        (normalPDF : numSigmas | uniformPDF)
    dataTable
        {character data of comma- or white space-separated table values}
```

`griddedTableDef` attributes:

`gtID`	An internal reference that is unique within the file.

`name`	A UNICODE name for the table (may be the same string as `gtID`).
`units`	The units-of-measure of the table's output signal. See Section 6.5.6 (p. 49) below.

`griddedTableDef` sub-elements:

`description`	The optional description element allows the author to describe the data contained within this `griddedTable`.
`provenance`	The optional provenance element allows the author to describe the source and history of the data within this `griddedTable`. Alternatively, a `provenanceRef` reference can be made to a previously defined `provenance`.
`breakpointRefs`	The mandatory `breakpointRefs` element contains separate `bpRef` elements, each pointing to a separately defined `breakpointDef`. Thus, the independent coordinates associated with this function table are defined elsewhere and only a reference is given here. The order of appearance of the `bpRef`s is important; see the text above.
`uncertainty`	This optional element, if present, describes the uncertainty of this parameter. See Section 6.4 (p. 42) for more information about this element.
`dataTable`	The numeric values of the function at the function vertices specified by the breakpoint sets are contained within this element, in a single comma- or white space-separated list, representing an unraveled multi-dimensional table. Parsing this list and storing it in the appropriate array representation is up to the implementer. By convention, the last breakpoint value increases most rapidly.

Example 8. An excerpt showing an example of a `griddedTableDef` element

This nonlinear function table is used by a subsequent `function` in Example 11 (p. 34) to specify an output value based on two input values: body flap deflection and Mach number. This table is defined outside of a `function` element because this particular function table is used by two functions: one for the left-lower body flap and one for the lower-right body flap; thus, their actual independent (input) variable values might be different.

```
      <!-- ================================== -->     ❶
      <!-- Lower Body Flap Tables (definitions) -->
      <!-- ================================== -->

 <griddedTableDef name="CLBFL0" gtID="CLBFL0_table">     ❷
   <description>      ❸
```

```
        Lower body flap contribution to lift coefficient,
        polynomial constant term
    </description>
    <provenance>        ❹
      <author name="Bruce Jackson" org="NASA Langley Research Center"
 email="e.b.jackson@larc.nasa.gov"/>
      <creationDate date="2003-01-31"/>
      <documentRef docID="REF01"/>
    </provenance>
    <breakpointRefs> ❺
      <bpRef bpID="DBFL_PTS"/>
      <bpRef bpID="XMACH1_PTS"/>
    </breakpointRefs>
    <dataTable> <!-- last breakpoint (XMACH) changes most rapidly --> ❻
<!-- CLBFL0 POINTS  -->
<!-- DBFL =        0.0       -->
 0.00000E+00 , 0.00000E+00 , 0.00000E+00 , 0.00000E+00 , 0.00000E+00 ,
 0.00000E+00 , 0.00000E+00 , 0.00000E+00 , 0.00000E+00 , 0.00000E+00 ,
 0.00000E+00 , 0.00000E+00 , 0.00000E+00 ,
<!-- DBFL =       15.0       --> ❼
-0.86429E-02 ,-0.10256E-01 ,-0.11189E-01 ,-0.12121E-01 ,-0.13520E-01 ,
-0.86299E-02 ,-0.53679E-02 , 0.76757E-02 , 0.11300E-01 , 0.62992E-02 ,
 0.51902E-02 , 0.38813E-02 , 0.37366E-02 ,
<!-- DBFL =       30.0       -->
 0.22251E-01 , 0.26405E-01 , 0.28805E-01 , 0.31206E-01 , 0.34806E-01 ,
 0.31321E-01 , 0.28996E-01 , 0.19698E-01 , 0.18808E-01 , 0.12755E-01 ,
 0.10804E-01 , 0.98493E-02 , 0.83719E-02 ,
<!-- DBFL =       45.0       -->
   .
   . [other points in table]
   .
    </dataTable>
  </griddedTableDef>
```

❶ Comments are good practice for human readers

❷ `name` is used for documentation purposes; `gtID` is intended for automatic wiring (autocode) tools.

❸ Descriptions make for good practice whenever possible. Here we explain the contents of the function represented by the data points.

❹ `provenance` is the story of the origin of the data.

❺ These `bpRef`s are in the same order as the table is wrapped (see text above) and must be reflected in the referencing function in the same order. In this excerpt, the referencing function must list the `independentVarRefs` such that the signal that represents delta body flap (`DBFL`) must precede the reference to the signal that represents Mach number (`XMACH`).

❻ The points listed within the `dataTable` element are given as if the last `bpRef` varies most rapidly. See the discussion above.

❼ Embedded comments are a good practice.

6.2.5. The ungridded table definition element

The `ungriddedTableDef` element defines a set of non-orthogonal data points, along with their independent values (coordinates), corresponding with the dependent value of an arbitrary function.

A 'non-orthogonal' data set, as opposed to a gridded or 'orthogonal' data set, means that the independent values are not laid out in an orthogonal grid. This form must be used if the dependent

coordinates in any table dimension cannot be expressed by a single monotonically increasing vector.

See the excerpts below for two instances of ungridded data.

An optional `uncertainty` element may be provided that represents the statistical variation in the values presented. See the section on Statistics below for more information about this element.

```
ungriddedTableDef : utID, [name, units]
    description? :
        {description character data}
    EITHER
        provenanceRef? : provID
    OR
        provenance? : [provID]
            author+ : name, org, [email]
                contactInfo* : [contactInfoType, contactLocation]
                    {text describing contact information}
            creationDate : date {in YYYY-MM-DD format}
            documentRef* : [docID,] refID
            modificationRef* : modID
            description?
    uncertainty? : effect
        (normalPDF : numSigmas) | (uniformPDF : bounds+)
    dataPoint+ :
        {coordinate/value sets as character data}
```

`ungriddedTableDef` attributes:

`utID`	An internal reference that is unique within the file
`name`	An optional UNICODE name for the table (may be the same string as `utID`).
`units`	Optional units-of-measure for the table's output signal.

`ungriddedTableDef` sub-elements:

`description`	The optional description element allows the author to describe the data contained within this `ungriddedTable`.
`provenance`	The optional provenance element allows the author to describe the source and history of the data within this `ungriddedTable`. Alternatively, a `provenanceRef` reference can be made to a previously defined provenance.
`uncertainty`	This optional element, if present, describes the uncertainty of this parameter. See the section on Statistics below for more information about this element.
`dataPoint`	One or more sets of coordinate and output numeric values of the function at various locations within its input space. This element includes one coordinate for each

function input variable. Parsing this information into a usable interpolative function is up to the implementer. By convention, the coordinates are listed in the same order that they appear in the parent function.

Example 9. An excerpt showing an `ungriddedTableDef` element, encoding the data depicted in Figure 2 (p. 29).

This 2D function table is an example provided by Dr. Peter Grant of the University of Toronto. Such a table definition would be used in a subsequent `function` to describe how an output variable would be defined based on two independent input variables. The function table does not indicate which input and output variables are represented; this information is supplied by the `function` element later so that a single function table can be reused by multiple functions.

```
 <ungriddedTableDef name="CLBASIC as function of flap angle and angle-of-
        attack" utID="CLBAlfaFlap_Table" units="nd">
   <description>
     CL basic as a function of flap angle and angle-of-attack. Note the alpha
     used in this table is with respect to the wing design plane (in degrees).
     Flap is in degrees as well.
   </description>

   <provenance>
     <author name="Peter Grant" org="UTIAS"/>  ❶
     <creationDate date="2006-11-01"/>
     <documentRef refID="PRG1" />
   </provenance>

   <!--For ungridded tables you provide a list of dataPoints--> ❷
        <dataPoint> 1.0 -5.00  -0.44 <!-- flap, alfawdp, CLB--></dataPoint>   ❸
        <dataPoint> 1.0  10.00  0.95 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 1.0  12.00  1.12 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 1.0  14.00  1.26 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 1.0  15.00  1.32 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 1.0  17.00  1.41 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 5.0 -5.00  -0.55 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 5.0  0.00  -0.03 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 5.0  5.00   0.50 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 5.0  10.00  1.02 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 5.0  12.00  1.23 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 5.0  14.00  1.44 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 5.0  16.00  1.63 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 5.0  17.00  1.70 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 5.0  18.00  1.75 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint modID='A'> 10.0 -5.00 -0.40 <!-- flap, alfawdp, CLB--></
dataPoint> ❹
        <dataPoint> 10.0 14.00  1.57 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 10.0 15.00  1.66 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 10.0 16.00  1.75 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 10.0 17.00  1.80 <!-- flap, alfawdp, CLB--></dataPoint>
        <dataPoint> 10.0 18.00  1.84 <!-- flap, alfawdp, CLB--></dataPoint>

</ungriddedTableDef>
```

❶ Example courtesy of Dr. Peter Grant, U. Toronto

❷ Comments are a good idea for human readers

❸ For a 2D table such as this one, data points give two columns of independent breakpoint values and a third column of function values at those breakpoints.

❹ The `modID` attribute implies this point was edited during modification 'A' of this model, as described in the file header information.

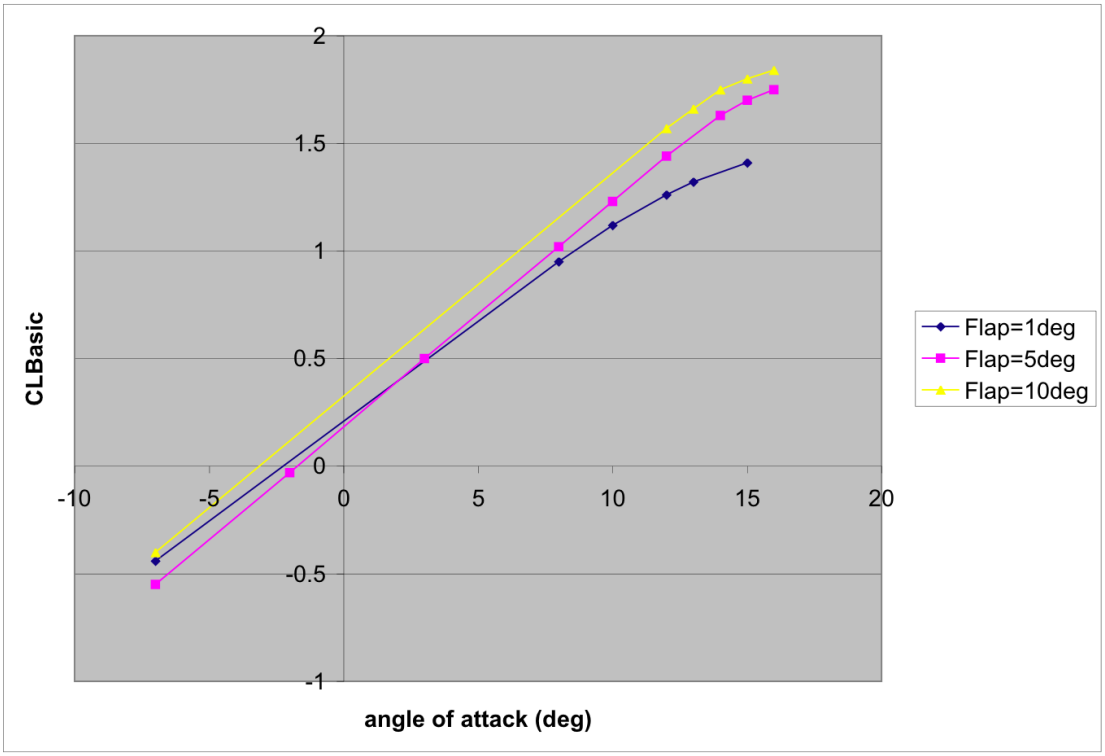

Figure 2. The 2D lift function given in Example 9 (p. 28)

Example 10. An excerpt from a sample aerodynamics model giving an example of a 3D `ungriddedTableDef` element, encoding the data shown in Figure 3 (p. 31).

In this example, the dependent coordinates all vary in each dimension.

```
    <!--==================================-->  ❶
    <!--  Three-D Table Definition Example -->
    <!--==================================-->

    <ungriddedTableDef name="yawMomentCoefficientTable1" units="nd"
utID="yawMomentCoefficientTable1">
              <!--  alpha,     beta,      delta:     yawMomentCoeff -->    ❷
      <dataPoint> -1.8330592 -5.3490387 -4.7258599 -0.00350641</dataPoint>
      <dataPoint> -1.9302179 -4.9698462 0.2798654  -0.0120538</dataPoint>
      <dataPoint> -2.1213095 -5.0383145 5.2146443  -0.0207089</dataPoint>
      <dataPoint> 0.2522004  -4.9587161 -5.2312860 -0.000882368</dataPoint>
      <dataPoint> 0.3368831  -5.0797159 -0.3370540 -0.0111846</dataPoint>
      <dataPoint> 0.2987289  -4.9691198 5.2868938  -0.0208758</dataPoint>
      <dataPoint> 1.8858257  -5.2077654 -4.7313074 -0.00219842</dataPoint>
      <dataPoint> 1.8031083  -4.7072954 0.0541231  -0.0111726</dataPoint>
      <dataPoint> 1.7773659  -5.0317988 5.1507477  -0.0208074</dataPoint>
      <dataPoint> 3.8104785  -5.2982162 -4.7152852 -0.00225906</dataPoint>
      <dataPoint> 4.2631596  -5.1695257 -0.1343410 -0.0116563</dataPoint>
      <dataPoint> 4.0470946  -5.2541017 5.0686926  -0.0215506</dataPoint>
      <dataPoint> -1.8882611 0.2422452  -5.1959304 0.0113462</dataPoint>
```

```
  <dataPoint> -2.1796091 0.0542085  0.2454711  -0.000253915</dataPoint>
  <dataPoint> -2.2699103 -0.3146657 4.8638859  -0.00875431</dataPoint>
  <dataPoint> 0.0148579  0.1095599  -4.9639500 0.0105144</dataPoint>
  <dataPoint> -0.1214591 -0.0047960 0.2788827  -0.000487753</dataPoint>
  <dataPoint> 0.0610233  0.2029588  5.0831767  -0.00816086</dataPoint>
  <dataPoint> 1.7593356  -0.0149007 -5.0494446 0.0106762</dataPoint>
  <dataPoint> 1.9717048  -0.0870861 0.0763833  -0.000332616</dataPoint>
  <dataPoint> 2.0228263  -0.2962294 5.1777078  -0.0093807</dataPoint>
  <dataPoint> 4.0567507  -0.2948622 -5.1002243 0.010196</dataPoint>
  <dataPoint> 3.6534822  0.2163747  0.1369900  0.000312733</dataPoint>
  <dataPoint> 3.6848003  0.0884533  4.8214805  -0.00809437</dataPoint>
  <dataPoint> -2.3347682 5.2288720  -4.7193014 0.02453</dataPoint>
  <dataPoint> -2.3060350 4.9652745  0.2324610  0.0133447</dataPoint>
  <dataPoint> -1.8675176 5.0754646  5.1169942  0.00556052</dataPoint>
  <dataPoint> 0.0004379  5.1220145  -5.2734993 0.0250468</dataPoint>
  <dataPoint> -0.1977035 4.7462188  0.0664495  0.0124083</dataPoint>
  <dataPoint> -0.1467742 5.0470092  5.1806131  0.00475277</dataPoint>
  <dataPoint> 1.6599338  4.9352809  -5.1210532 0.0242646</dataPoint>
  <dataPoint> 2.2719825  4.8865093  0.0315210  0.0125658</dataPoint>
  <dataPoint> 2.0406858  5.3253471  5.2880688  0.00491779</dataPoint>
  <dataPoint> 4.0179983  5.0826426  -4.9597629 0.0243518</dataPoint>
  <dataPoint> 4.2863811  4.8806558  -0.2877697 0.0128886</dataPoint>
  <dataPoint> 3.9289361  5.2246849  4.9758705  0.00471241</dataPoint>
  <dataPoint> -2.2809763 9.9844584  -4.8800790 0.0386951</dataPoint>
  <dataPoint> -2.0733070 9.9204337  0.0241722  0.027546</dataPoint>
  <dataPoint> -1.7624546 9.9153493  5.1985794  0.0188357</dataPoint>
  <dataPoint> 0.2279962  9.8962508  -4.7811258 0.0375762</dataPoint>
  <dataPoint> -0.2800363 10.3004593 0.1413907  0.028144</dataPoint>
  <dataPoint> 0.0828562  9.9008011  5.2962722  0.0179398</dataPoint>
  <dataPoint> 1.8262230  10.0939436 -4.6710211 0.037712</dataPoint>
  <dataPoint> 1.7762123  10.1556398 -0.1307093 0.0278079</dataPoint>
  <dataPoint> 2.2258599  9.8009720  4.6721747  0.018244</dataPoint>
  <dataPoint> 3.7892651  9.8017197  -4.8026383 0.0368199</dataPoint>
  <dataPoint> 4.0150716  9.6815531  -0.0630955 0.0252014</dataPoint>
  <dataPoint> 4.1677953  9.8754433  5.1776223  0.0164312</dataPoint>
</ungriddedTableDef>
```

❶ Example courtesy of Mr. Geoff Brian, DSTO

❷ In this example, columns are labeled with an XML comment for human readers. Actual association of each input (alpha, beta and delta) and the single output (yawing moment) to the respective input and output signals is mechanized in any subsequent function definition(s).

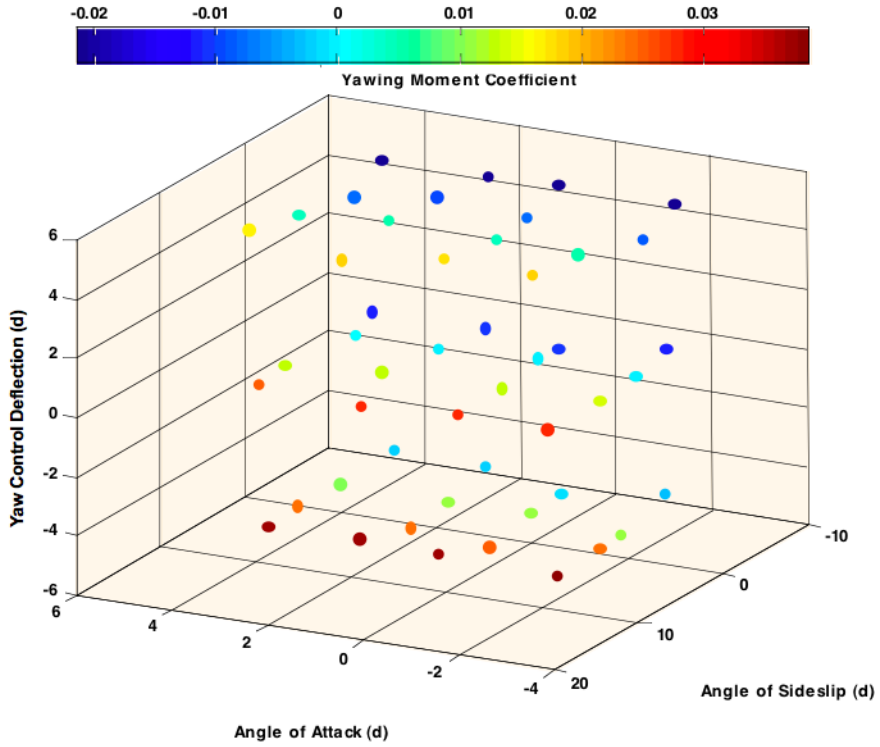

Figure 3. The 3D function given in the previous example

6.2.6. The function definition element

The `function` element connects breakpoint sets (for gridded tables), independent variables, and data tables (gridded or ungridded) to their respective output variable.

```
function : name
    description? :
        {text description of the function}
    EITHER
        provenanceRef? : provID
    OR
        provenance? : [provID]
            author+ : name, org, [email]
                contactInfo* : [contactInfoType, contactLocation]
                    {text describing contact information}
            creationDate : date {in YYYY-MM-DD format}
            documentRef* : [docID,] refID
            modificationRef* : modID
            description?
    EITHER
    (
      independentVarPts+ : varID, [name, units, sign, extrapolate, interpolate]
          {input values as character data}
      dependentVarPts : varID, [name, units, sign]
          {output values as character data}
    )
    OR
    (
```

```
        independentVarRef+ : varID, [min, max, extrapolate, interpolate]
        dependentVarRef : varID
        functionDefn : [name]
            CHOICE OF
            (
                griddedTableRef : gtID
            OR
                griddedTableDef : gtID, [name, units]
                  description?
                      {text description of table}
                  (provenance | provenanceRef)?   {as described earlier}
                  breakpointRefs
                      bpRef+ : bpID
                  uncertainty? : effect
                      (normalPDF : numSigmas) | (uniformPDF : bounds+)
                  dataTable
                      {gridded data table as character data}
            OR
                ungriddedTableRef : utID
            OR
                ungriddedTableDef : utID, [name, units]
                  description?
                      {text description of table}
                  (provenance | provenanceRef)?   {as described earlier}
                  uncertainty? : effect
                      (normalPDF : numSigmas) | (uniformPDF : bounds+)
                  dataPoint+
                      {coordinate/value sets as character data}
            )
    )
```

`function` attributes:

`name`	A UNICODE name for the function.

`function` sub-elements:

`description`	The optional description element allows the author to describe the data contained within this `function`.
`provenance`	The optional provenance element allows the author to describe the source and history of the data within this `function`. Alternatively, a `provenanceRef` reference can be made to a previously defined provenance.
`independentVarPts`	If the author chooses, she can express a linearly interpolated functions by specifying the independent (breakpoint) value sets as one or more `independentVarPts` which are comma- or white space-separated, monotonically increasing floating-point coordinate values corresponding to the `dependentVarPts`. In the case of multiple dimensions, more than one `independentVarPts` must be specified, one for each dimension. The mandatory `varID` attribute is used to connect each `independentVarPts` set with an input variable.

An optional `interpolate` attribute specifies the preference for using linear, quadratic, or cubic relaxed splines for calculating dependent values when the independent arguments are in between specified values. When not specified, the expectation would be to use a linear spline interpolation between points. The performance of interpolation of various orders is left up to the processing application. See Section 6.3 (p. 38).

`dependentVarPts` — This element goes along with the previous element to specify a function table. Only one `dependentVarPts` may be specified. If the function is multi-dimensional, the convention is the last breakpoint dimension changes most rapidly in this comma- or white space-separated list of floating-point output values. The mandatory `varID` attribute is used to connect this table's output to an output variable.

`independentVarRef` — One or more of these elements refers to separately defined `variableDef`s. The order of specification is important and must match the order in which breakpoint sets are specified or the order of coordinates in ungridded table coordinate/value sets.

An optional `interpolate` attribute specifies the preference for using discrete, linear, quadratic, or cubic splines for calculating dependent values when the independent arguments are in between specified values. When not specified, the default expectation would be a linear spline interpolation between points. The performance of interpolation of various orders is left up to the implementer. See Section 6.3 (p. 38).

`dependentVarRef` — A single `dependentVarRef` must be specified to connect the output of this function to a particular `variableDef`.

`functionDefn` — This element identifies either a separately specified data table definition or specifies a private table, either gridded or ungridded.

`griddedTableRef` — If not defining a simple function table, the author may use this element to point to a separately specified `griddedTableDef` element.

`ungriddedTableRef` — If not using a simple function table, the author may use this element to point to separately specified `ungriddedTableDef` element.

Example 11. An excerpt giving the example of a function which refers to a previously defined `griddedTableDef`

This example ties the input variables `DBFLL` and `XMACH` into output variable `CLBFLLO` through a function called `CLBFLO_fn`, which is represented by the linear interpolation of the gridded table previously defined by the `CLBFL0_table` `griddedTableDef` (see the griddedTableDef example above).

```
      <!-- ============================== -->
      <!-- Lower left body flap functions -->
      <!-- ============================== -->

  <function name="CLBFLL0">
    <description>
      Lower left body flap lookup function for lift, polynomial constant term.
    </description>
    <independentVarRef varID="DBFLL"  min="0.0" max="60." extrapolate="neither"/> ❶
    <independentVarRef varID="XMACH" min="0.3" max="4.0" extrapolate="neither"/>
    <dependentVarRef varID="CLBFLL0"/> ❷
    <functionDefn name="CLBFL0_fn">
      <griddedTableRef gtID="CLBFL0_table"/> ❸
    </functionDefn>
  </function>
```

❶ The independent variables must be given in the order of least-rapidly changing to most-rapidly changing values in the previously defined function table. The processing application needs to pay attention to the `extrapolate` attribute, which specifies how to treat a variable whose value exceeds the stated limits on input. See Section 6.3 (p. 38).

❷ The dependent variable (identified as `CLBFLL0`) is the output variable for this function. `CLBFLL0` must have been declared previously with a `variableDef` element.

❸ This is a reference to the previously declared `griddedTableDef`.

Example 12. A `function` with an internal table

In this example, the function `CLRUD0` returns, in the variable `CLRUD0`, the value of function `CLRUD0_fn` represented by gridded `CLRUD0_table`. The inputs to the function are `abs_rud` and `XMACH` which are used to normalize breakpoint sets `DRUD_PTS` and `XMACH1_PTS` respectively. The input variables are limited between 0.0 to 30.0 and 0.3 to 4.0, respectively.

```
      <!-- ================ -->
      <!-- Rudder functions -->
      <!-- ================ -->

<!-- The rudder functions are only used once, so their table
     definitions are internal to the function definition.
--> ❶

  <function name="CLRUD0_fn"> ❷
    <description>
        Rudder contribution to lift coefficient,
        polynomial multiplier for constant term.
    </description>
    <provenance> ❸
      <author name="Bruce Jackson" org="NASA Langley Research Center"
 email="e.b.jackson@larc.nasa.gov"/>
```

```
      <creationDate date="2003-01-31"/>
      <documentRef docID="REF01"/>
    </provenance>
    <independentVarRef varID="abs_rud"  min="0.0" max="30." extrapolate="neither"/>
    <independentVarRef varID="XMACH" min="0.3" max="4.0" extrapolate="neither"/>
    <dependentVarRef varID="CLRUD0"/> ❹

    <functionDefn name="CLRUD0_fn_defn">
      <griddedTableDef name="CLRUD0_table"> ❺
        <breakpointRefs>
          <bpRef bpID="DRUD_PTS"/>
          <bpRef bpID="XMACH1_PTS"/>
        </breakpointRefs>
        <dataTable> <!-- last breakpoint changes most rapidly -->
<!-- CLRUD0  POINTS  -->
<!-- RUD =    0.0    -->
 0.00000E+00 , 0.00000E+00 , 0.00000E+00 , 0.00000E+00 , 0.00000E+00 ,
 0.00000E+00 , 0.00000E+00 , 0.00000E+00 , 0.00000E+00 , 0.00000E+00 ,
 0.00000E+00 , 0.00000E+00 , 0.00000E+00 ,
<!-- RUD =   15.0    -->
-0.13646E-01 , 0.26486E-01 , 0.16977E-01 ,-0.16891E-01 , 0.10682E-01 ,
 0.75071E-02 , 0.53891E-02 ,-0.30802E-02 ,-0.59013E-02 ,-0.95733E-02 ,
 0.00000E+00 , 0.00000E+00 , 0.00000E+00 ,
<!-- RUD =   30.0    -->
-0.12709E-02 , 0.52971E-01 , 0.33953E-01 ,-0.33782E-01 , 0.21364E-01 ,
 0.15014E-01 , 0.10778E-01 ,-0.61604E-02 ,-0.11803E-01 ,-0.19147E-01 ,
 0.00000E+00 , 0.00000E+00 , 0.00000E+00
        </dataTable>
      </griddedTable>
    </functionDefn>
  </function>
```

❶ This comment helps humans understand the reason for an embedded table.

❷ The `name` attribute is used for documentation purposes, not for internal variable linkage.

❸ The `provenance` element is required by the AIAA standard [AIAA11].

❹ The `varID` attribute is used to link the output of this function with a previously defined variable as given in Example 3 (p. 19).

❺ This example has an embedded gridded table.

Example 13. A simple 1D `function`

At the other end of the spectrum, a simple 1D nonlinear function can be defined with no reuse, as shown below; however, multiple-dimension functions are supported by adding additional `independentVarPts` definitions.

```
<function name="CL">
  <independentVarPts varID="alpdeg"> ❶
   -4.0, 0., 4.0, 8.0, 12.0, 16.0
  </independentVarPts>
  <dependentVarPts varID="cl"> ❷
    0.0, 0.2, 0.4, 0.8, 1.0, 1.2
  </dependentVarPts>
</function>
```

❶ Breakpoints in angle-of-attack are listed here.

❷ Values of `cl` are given, corresponding to the angle-of-attack breakpoints given previously.

6.2.7. The checkData element

The `checkData` element contains one or more input/output vector pairs (and optionally a dump of internal values) for the encoded model to assist in verification and debugging of the implementation.

```
checkData
  staticShot+ : name, [refID]
        description?
            {text description of the static test case}
                (provenance | provenanceRef)?   {as defined previously}
        checkInputs
            signal+
                signalName
                signalUnits
                signalValue
        internalValues?
            signal+
                varID
                signalValue
        checkOutputs
            signal+
                signalName
                signalUnits
                signalValue
                tol
```

`checkData` sub-elements:

`staticShot`	One or more check-case data sets, each of which contain mandatory sub-elements `checkInputs` and `checkOutputs` vectors (with required match tolerances), and optional `provenance`, `provenanceRef`, `description` and `internalValues` sub-elements.

Example 14. Check-case data set excerpt

A DAVE-ML file excerpt specifying a check-case data set example for a simple model

```
<checkData>
  <staticShot name="Nominal" refID="NOTE1"> ❶
    <description>An example static check of a simple DAVE-ML model</description>
    <checkInputs>  ❷
      <signal>  ❸
        <signalName>trueAirspeed</signalName>
        <signalUnits>f_s</signalUnits>
        <signalValue> 300.000</signalValue>
      </signal>
      <signal>
        <signalName>angleOfAttack</signalName>
        <signalUnits>deg</signalUnits>
        <signalValue>   5.000</signalValue>
      </signal>
       .
       .       (similar input signals omitted)
       .
      <signal>
        <signalName>delta elevator</signalName>
```

```
        <signalUnits>deg</signalUnits>
        <signalValue>   0.000</signalValue>
      </signal>
    </checkInputs>
    <checkOutputs> ❹
      <signal>  ❺
        <signalName>CX</signalName>
        <signalUnits>nd</signalUnits>
        <signalValue>-0.00400000000000</signalValue>
        <tol>0.000001</tol>
      </signal>
       .
       .       (similar output signals omitted)
       .
    </checkOutputs>
  </staticShot>
  <staticShot name="Positive pitch rate"> ❻
    <checkInputs>
       .
       .       (similar input and output signal information omitted)
       .
    </checkOutputs>
  </staticShot>
  <staticShot name="Positive elevator">
    <checkInputs>
       .
       .       (similar input and output signal information omitted)
       .
    </checkOutputs>
  </staticShot>
</checkData>
```

❶ This first check-case refers to a note given in the file header; this is useful to document the source of the check-case values.

❷ The `checkInputs` element defines the input variable values, by variable name, as well as units (so they can be verified).

❸ Multiple `signal` elements are usually given; taken together, these scalar `signals`s represent the check-case input "vector."

❹ The `checkOutputs` element defines output variable values that should result from the specified input values.

❺ Note the included tolerance value, indicating the absolute value tolerance within which the output values must match the check-case data values.

❻ Multiple check-cases may be specified; this one differs from the previous check-case due to an increase in the pitching-rate input.

Example 15. A second `checkData` example with internal values

This example shows another check-case; this one includes intermediate values as an aide to debugging a new implementation.

```
 <checkData>
   <( provenance | provenanceRef )?>
   <staticShot name="Skewed inputs">
     <description>
       Another example static check; this one includes all the internal, intermediate
calculations
       to assist in debugging the implementation.
     </description>
     <checkInputs>
       <signal>
```

```
        <signalName>trueAirspeed</signalName>
        <signalUnits>f_s</signalUnits>\
        <signalValue> 300.000</signalValue>
      </signal>
      <signal>
        <signalName>angleOfAttack</signalName>
        <signalUnits>deg</signalUnits>
        <signalValue>  16.200</signalValue>
      </signal>
       .
       .       (similar input values omitted)
       .
      <signal>
        <signalName>bodyPositionOfCmWrtMrc</signalName>
        <signalUnits>fracMAC</signalUnits>
        <signalValue>   0.123</signalValue>
      </signal>
    </checkInputs>
    <internalValues> ❶
      <signal>
        <varID>vt</varID>
        <signalValue>300.0</signalValue>
      </signal>
      <signal>
        <varID>alpha</varID>
        <signalValue>16.2</signalValue>
      </signal>
       .
       .       (similar internal values omitted)
       .
    </internalValues>
    <checkOutputs>
      <signal>
        <signalName>aeroBodyForceCoefficient_X</signalName>
        <signalValue> 0.04794994533333</signalValue>
        <signalUnits>nd</signalUnits>
        <tol>0.000001</tol>
      </signal>
      <signal>
        <signalName>aeroBodyForceCoefficient_Z</signalName>
        <signalValue>-0.72934852554344</signalValue>
        <signalUnits>nd</signalUnits>
        <tol>0.000001</tol>
      </signal>
      <signal>
        <signalName>aeroBodyMomentCoefficient_Pitch</signalName>
        <signalValue>-0.10638585796503</signalValue>
        <signalUnits>nd</signalUnits>
        <tol>0.000001</tol>
      </signal>
    </checkOutputs>
  </staticShot>
</checkData>
```

❶ The `internalValues` element, if present, usually is a list of all model-defined internal variable values. This is normally not required for a model exchange, but such information is very useful to aide in debugging the implementation by the recipient.

6.3. Function interpolation/extrapolation

It is possible to specify the method of interpolation to be used for nonlinear function tables by use of the `interpolate` attribute of the `independentVarPts` and `independentVarRef`

elements. This attribute, combined with the `extrapolate` flag, provides several different ways of realizing the intermediate values of the function when not at one of the specified intersections of independent values.

Possible values for the `interpolate` attribute are:

`discrete`	Output uses value associated with nearest breakpoint
`floor`	Output uses value associated with next (numerically higher) breakpoint
`ceiling`	Output uses value associated with last (numerically lower) breakpoint
`linear` (default)	Output is linearly interpolated between breakpoints
`quadraticSpline`	Output follows a quadratic spline fit through values associated with two nearby breakpoints
`cubicSpline`	Output follows a cubic spline fit through values associated with three nearby breakpoints

Possible values for the `extrapolate` attribute are:

`neither` (default)	Output is held constant at value associated with closest end of breakpoints if the input value is outside the limits of the associated breakpoint set
`min`	Output follows extrapolated values of function if the input is below the minimum breakpoint value
`max`	Output follows extrapolated values of function if the input is above maximum breakpoint value
`both`	Output follows extrapolated values of function if the input is outside the limits of the associated breakpoint set

Implementation of the specific interpolation algorithm is left up to the implementer. One reference is the Wikipedia entry on interpolation [wiki01].

The following implementation notes are suggested:

- An infinite set of quadratic interpolations are possible; it is suggested to use the one that minimizes either the deviation from a linear interpolation or the slope error at any edge.
- For cubic interpolation, the natural cubic spline (which has a second derivative of zero at each end) is recommended when the `extrapolate` attribute is `none`. When the `extrapolate` attribute is `both`, a clamped cubic spline that matches the extrapolated slope of the last two data points is suggested.
- For the discrete interpolation values (`discrete`, `ceiling`, or `floor`), the value of the `extrapolate` attribute is meaningless.

For discrete interpolation,

- `discrete` implies the change between output values occurs midway between independent breakpoint values, as shown in the top plot of Figure 4 (p. 41).

- `ceiling` means the output takes on the value of the next-higher dependent variable breakpoint as soon as each independent breakpoint value is passed (assuming the input value is increasing) as shown in the middle plot of Figure 4 (p. 41).

- `floor` means the output retains the value of the last dependent variable breakpoint until the next independent breakpoint value is reached (assuming the input value is increasing) as shown in the bottom plot of Figure 4 (p. 41).

The default value for `interpolate` is `linear`. The default value for `extrapolate` is `neither`.

Figures 4 (p. 41)and 5 (p. 42)below give nine different examples for a 1D table whose independent values are [1, 3, 4, 6, 7.5] with dependent values of [2, 6, 5, 7, 1.5].

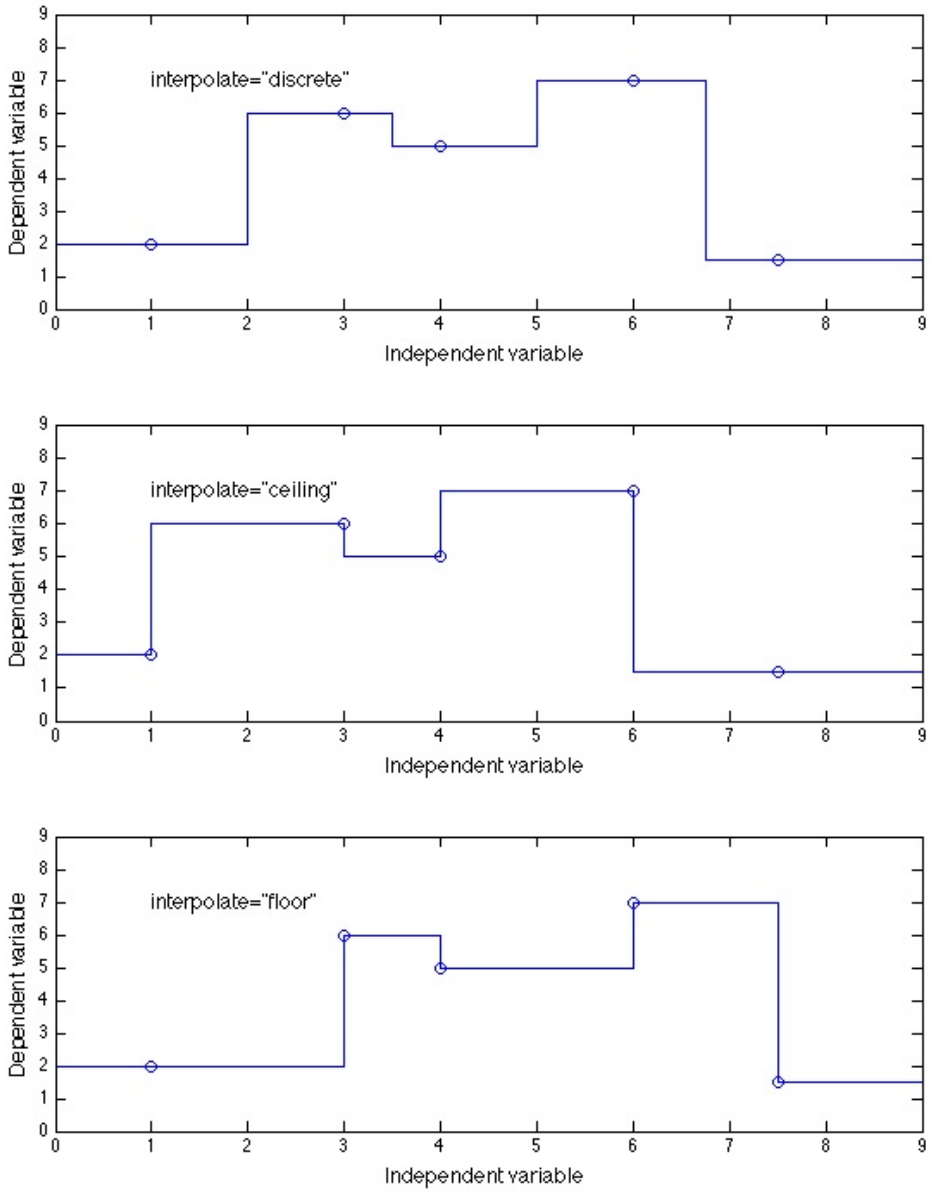

Figure 4. Example of the three discrete enumeration values of `interpolate` attribute of the `independentVarPts` and `independentVarRef` elements for a 1D function table.

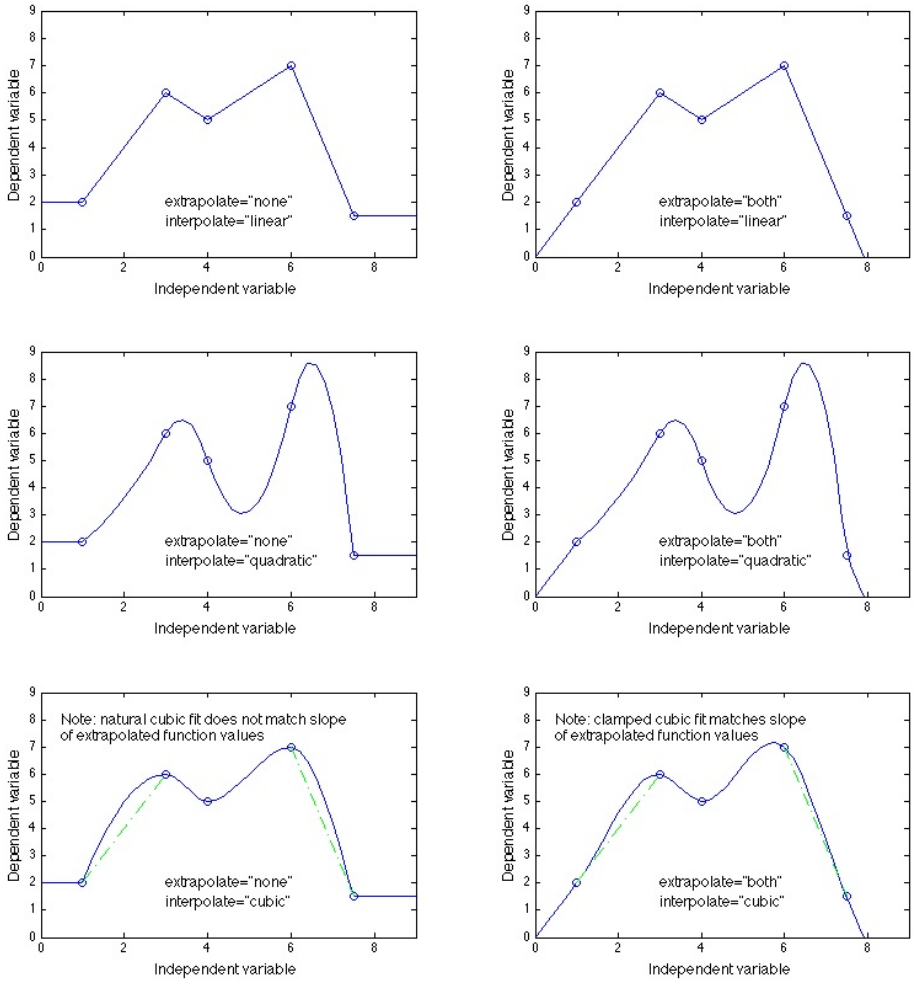

Figure 5. Examples of the three higher-order interpolation methods showing the effect of the `interpolate` attribute of the `independentVarPts` and `independentVarRef` elements for a 1D function table.

6.4. Statistical information encoding

Statistical measures of variation of certain parameters and functions can be embedded in a DAVE-ML model in several ways. This information is captured in an `uncertainty` element, which can be referenced by `variableDef`, `griddedTableDef` and `ungriddedTableDef` elements. For maximum modeling flexibility, it is possible to add uncertainty to the independent value arguments to a function or calculation, to the output of a function itself, as well as to any output signal. Applying uncertainty at more than one location in a calculation change is probably not a good practice, however.

Details on providing the random values for uncertainties is left to the implementer.

Uncertainty in the value of a parameter or function is given in one of two ways, depending on the appropriate probability distribution function (PDF): as a Gaussian or normal distribution (bell curve) or as a uniform (evenly spread) distribution. One of these distributions is selected by including either a `normalPDF` or a `uniformPDF` element within the `uncertainty` element.

Linear correlation between the randomness of two or more variables or functions can be specified. Although the correlation between parameters does not have a dependency direction (i.e., the statistical uncertainty of one parameter is specified in terms of the other parameter, therefore the calculation order does not matter), correlation is customarily specified as a dependency of one random variable on the value of another random variable. `correlatesWith` identifies variables or functions whose uncertainty 'depends' on the current value of this variable or parameter; the `correlation` sub-element specifies the correlation coefficient and identifies the (previously calculated) random variable or function on which the correlation depends.

These correlation sub-elements only apply to normal (Gaussian) probability distribution functions.

Each of these distribution description elements contain additional information, as described below.

```
uncertainty : effect=['additive'|'multiplicative'|'percentage'|'absolute']
  EITHER
    normalPDF : numSigmas=['1'|'2'|'3']
        bounds { scalar value representing the one, two or three sigma bound }:
        (correlatesWith* : varID |
         correlation* : varID, corrCoef )
  OR
    uniformPDF
        bounds { one or two scalar values for abs. or min/max bounds }
```

`uncertainty` attributes:

`effect`	Indicates, by choice of four enumerated values, how the uncertainty is modeled: as an additive, multiplicative, or percentage variation from the nominal value, or a specific number (absolute).

`uncertainty` sub-elements:

`normalPDF`	If present, the uncertainty in the parameter value has a probability distribution that is Gaussian (bell-shaped). A single parameter representing the additive (± some value), percentage (± some %) of variation from the nominal value in terms of 1, 2, 3, or more standard deviations (sigmas) is specified. Note: multiplicative and absolute bounds do not make much sense.
`uniformPDF`	If present, the uncertainty in the parameter or function value has a uniform likelihood of taking on any value between symmetric or asymmetric boundaries, which

are specified in terms of additive (either ±x or +x/-y), multiplicative, percentage, or absolute variations. If absolute, the specified range of values must bracket the nominal value. For this element, the `bounds` sub-element may contain one or two values, in which case the boundaries are symmetric or asymmetric.

Example 16. A variable with absolute uncertainty bounds

This example shows how to specify that a constant parameter can take on a specified range of values with uniform probability distribution. The nominal value of the minimum drag coefficient is specified to be 0.005, but when performing parametric variations, it is allowed to take on values between 0.001 and 0.01.

```
<DAVEfunc>
    <fileHeader>
      .
      .
      .
    </fileHeader>
    <variableDef name="CD zero" varID="CDo" units="nd" initialValue="0.005">     ❶
        <description>
            Minimum coefficient of drag with an
            asymmetric uniform uncertainty band
        </description>
        <isOutput/>
        <uncertainty effect="absolute">    ❷
            <uniformPDF>
                <bounds>0.001</bounds>
                <bounds>0.010</bounds>
            </uniformPDF>
        </uncertainty>
    </variableDef>
 </DAVEfunc>
```

❶ We declare the parameter `CDo` as having a nominal value of 0.005.

❷ When parametric variations are applied, the value of `CDo` can vary uniformly between 0.001 and 0.010.

Example 17. 10% uncertainty applied to output variable with a uniform distribution

This example shows how to specify that a variable has a 10% uniformly distributed uncertainty band. In this example, the output variable comes from a nonlinear 1D function and the uncertainty is applied to the output of the table.

```
<DAVEfunc>
    <fileHeader>
       .
       .
       .
    </fileHeader>
    <variableDef name="angleOfAttack" varID="Alpha_deg" units="deg">
        <isStdAIAA/>
    </variableDef>
    <variableDef name="Cm_u" varID="Cm_u" units="nd">
        <description>
```

```
            Coefficient of pitching moment with 10 percent
            symmetric uniform uncertainty band
        </description>
        <isOutput/>
        <uncertainty effect="percentage">    ❶
            <uniformPDF>
                <bounds>10.0</bounds>
            </uniformPDF>
        </uncertainty>
    </variableDef>
    <breakpointDef bpID="ALP">
        <bpVals>0, 5, 10, 15, 20, 25, 30, 35</bpVals>
    </breakpointDef>
    <function name="Nominal Cm">
        <description>
            Nominal pitching moment values prior to application
            of uncertainty
        </description>
        <independentVarRef varID="Alpha_deg"/>
        <dependentVarRef varID="Cm_u"/>
        <functionDefn>   ❷
            <griddedTableDef>
                <breakpointRefs>
                    <bpRef bpID="ALP"/>
                </breakpointRefs>
                <dataTable>
5.2, 4.3, 3.1, 1.8, 0.3, 0.1, 0.0, -0.1
                </dataTable>
            </griddedTableDef>
        </functionDefn>
    </function>
</DAVEfunc>
```

❶ We declare the output variable `Cm_u` as having the uncertainty of ±10% uniform distribution.

❷ This function gives the nominal values of `Cm_u` as a 1D function of angle-of-attack (alpha).

Example 18. Asymmetric additive uncertainty applied to output variable with uniform distribution

This example shows how to specify that a variable has an asymmetric, uniformly distributed, additive uncertainty band. In this example, the output variable comes from a nonlinear 1D function and the uncertainty is applied to the output of the table.

```
<DAVEfunc>
    <fileHeader>
      .
      .
      .
    </fileHeader>
    <variableDef name="angleOfAttack" varID="Alpha_deg" units="deg">
        <isStdAIAA/>
    </variableDef>
    <variableDef name="Cm_u" varID="Cm_u" units="nd">
        <description>
            Coefficient of pitching moment with an
            asymmetric uniform uncertainty band
        </description>
        <isOutput/>
        <uncertainty effect="additive">  ❶
            <uniformPDF>
                <bounds>-0.50</bounds>
                <bounds>0.00</bounds>
```

```
            </uniformPDF>
        </uncertainty>
    </variableDef>
    <breakpointDef bpID="ALP">
        <bpVals>0, 5, 10, 15, 20, 25, 30, 35</bpVals>
    </breakpointDef>
    <function name="Nominal Cm">
        <description>
            Nominal pitching moment values prior to application
            of uncertainty
        </description>
        <independentVarRef varID="Alpha_deg"/>
        <dependentVarRef varID="Cm_u"/>   ❷
        <functionDefn>
            <griddedTableDef>
                <breakpointRefs>
                    <bpRef bpID="ALP"/>
                </breakpointRefs>
                <dataTable>
5.2, 4.3, 3.1, 1.8, 0.3, 0.1, 0.0, -0.1
                </dataTable>
            </griddedTableDef>
        </functionDefn>
    </function>
</DAVEfunc>
```

❶ We declare the output variable `Cm_u` varies by as much as #0.5 to +0.0 about the nominal value. This delta value is in the same units as the nominal value (i.e. it is not a multiplier or percentage but an additive delta to the nominal value which comes from the 1D `Cm_u` function table description).

❷ This function gives the nominal values of `Cm_u` as a 1D function of angle-of-attack (alpha).

Example 19. A 1D point-by-point, Gaussian distribution function

In this example, a Gaussian (normal) distribution function is applied to *each* point in a 1D function table, with the 3-sigma value expressed as a multiplier of the nominal value.

```
<DAVEfunc>
    <fileHeader>
       .
       .
       .
    </fileHeader>
    <variableDef name="angleOfAttack" varID="Alpha_deg" units="deg">
        <isStdAIAA/>
    </variableDef>
    <variableDef name="Cm_u" varID="Cm_u" units="nd">
        <description>
            Coefficient of pitching moment with 10 percent
            symmetric uniform uncertainty band
        </description>
        <isOutput/>
    </variableDef>
    <breakpointDef bpID="ALP">
        <bpVals>0, 5, 10, 15, 20, 25, 30, 35</bpVals>
    </breakpointDef>
  <function name="Uncertain Cm">
    <independentVarRef varID="Alpha_deg"/>
    <dependentVarRef varID="Cm_u"/>
    <functionDefn>
      <griddedTableDef>
        <breakpointRefs>
```

```
          <bpRef bpID="ALP"/>
        </breakpointRefs>
        <uncertainty effect="multiplicative">  ❶
          <normalPDF numSigmas="3">  ❷
            <bounds>
              <dataTable>  ❸
                0.10, 0.08, 0.06, 0.05, 0.05, 0.06, 0.07, 0.12
              </dataTable>
            </bounds>
          </normalPDF>
        </uncertainty>
        <dataTable>   ❹
          5.2, 4.3, 3.1, 1.8, 0.3, 0.1, 0.0, -0.1
        </dataTable>
      </griddedTableDef>
    </functionDefn>
  </function>
</DAVEfunc>
```

❶ This declares the statistical uncertainty bounds of the `Cm_u` dependent variable will be expressed as a multiplication of the nominal value.

❷ This declares that the probability distribution is a normal distribution and the bounds represent 3-sigma (99.7%) confidence bounds.

❸ This table lists three-sigma bounds of each point of the `Cm_u` function as a table. The table must have the same dimensions and independent variable arguments as the nominal function; it is in effect an overlay to the nominal function table, but the values represent the bounds as multiples of the nominal function value.

❹ This table defines the nominal values of the function; these values will be used if the random variable associated with the uncertainty of this function is zero or undefined by the application.

Example 20. Two nonlinear functions with correlated uncertainty

In this example, uncertainty in pitching-moment coefficient varies in direct correlation with lift coefficient uncertainty.

```
<DAVEfunc>
    <fileHeader> . . . </fileHeader>
    <variableDef name="angleOfAttack" varID="Alpha_deg" units="deg">
        <isStdAIAA/>
    </variableDef>
    <variableDef name="CL_u" varID="CL_u" units="nd">
        <description> Coefficient of lift with a symmetric Gaussian uncertainty
            of 20%; correlates with Cm uncertainty. </description>
        <uncertainty effect="multiplicative">    ❶
            <normalPDF numSigmas="3">
                <bounds>0.20</bounds>
                <correlatesWith varID="Cm_u"/>   ❷
            </normalPDF>
        </uncertainty>
    </variableDef>
    <variableDef name="Cm_u" varID="Cm_u" units="nd">
        <description> Coefficient of pitching moment with a symmetric Gaussian
 uncertainty
            distribution of 30%; correlates directly with lift uncertainty. </
description>
        <isOutput/>
        <uncertainty effect="percentage">        ❸
```

```
            <normalPDF numSigmas="3">
                <bounds>30</bounds>
                <correlation varID="CL_u" corrCoef="1.0"/>  ❹
            </normalPDF>
        </uncertainty>
    </variableDef>
    <breakpointDef bpID="ALP">
        <bpVals>0, 5, 10, 15, 20, 25, 30, 35</bpVals>
    </breakpointDef>
    <function name="Nominal CL">
        <description> Nominal lift coefficient values prior to uncertainty </
description>
        <independentVarRef varID="Alpha_deg"/>
        <dependentVarRef varID="CL_u"/>
        <functionDefn>
            <griddedTableDef>
                <breakpointRefs><bpRef bpID="ALP"/></breakpointRefs>
                <dataTable> 0.0, 0.1, 0.2, 0.3, 0.4, 0.45, 0.5, 0.45 </dataTable>
            </griddedTableDef>
        </functionDefn>
    </function>
    <function name="Nominal Cm">
        <description> Nominal pitching moment values prior to uncertainty </
description>
        <independentVarRef varID="Alpha_deg"/>
        <dependentVarRef varID="Cm_u"/>
        <functionDefn>
            <griddedTableDef>
                <breakpointRefs><bpRef bpID="ALP"/></breakpointRefs>
                <dataTable> 5.2, 4.3, 3.1, 1.8, 0.3, 0.1, 0.0, -0.1 </dataTable>
            </griddedTableDef>
        </functionDefn>
    </function>
</DAVEfunc>
```

❶ Lift coefficient has a nominal value that varies with angle-of-attack according to a nonlinear 1D table (given in the "Nominal CL" table defined in this example). When performing parametric variations, `CL_u` can take on a multiplicative variation of up to 20% (3-sigma) with a Gaussian distribution.

❷ This element indicates that the variation of lift coefficient correlates directly with the variation in pitching moment coefficient.

❸ Pitching-moment coefficient has a nominal value that varies as a function of angle-of-attack, according to a nonlinear 1D table (given in the "Nominal Cm" table defined in this example). When performing parametric variations, `Cm_u` can take on a 30% variation (3-sigma) with a Gaussian distribution.

❹ This element indicates that the variation of pitching moment correlates directly with the variation in lift coefficient.

6.5. Additional DAVE-ML conventions

To facilitate the interpretation of DAVE-ML packages, the following conventions are proposed. Failure to follow any of these conventions should be noted prominently in the data files and any cover documentation.

6.5.1. Ordering of points

In listing data points in multi-dimensional table definitions, the sequence should be such that the last dimension changes most rapidly. In other words, a table that is a function of *f(a,b,c)* should

list the value for *f(1,1,1)*, then *f(1,1,2)*, etc. This may be different than, say, a Fortran DATA, Matlab® script or C++ initialization statement; the responsibility for mapping the data in the correct sequence in the model realization is left up to the implementer.

Figure 6 (p. 49) below shows how a 3D table is represented as an unraveled list of points.

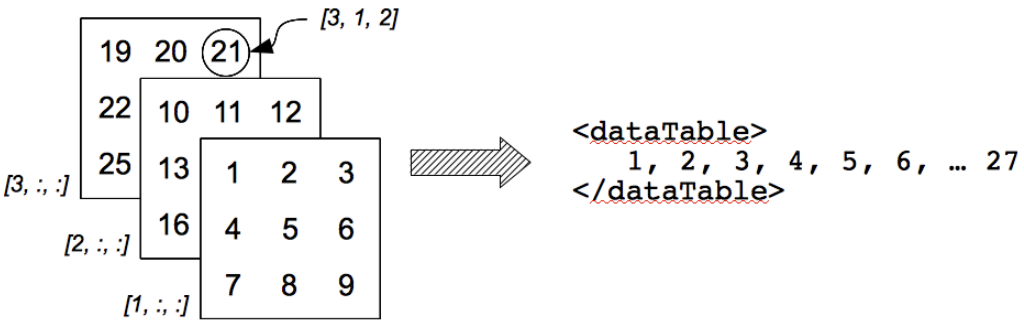

Figure 6. An unraveled list compared to the original 3D table

6.5.2. Locus of action of moments

It is recommended that all force and moments be considered to act around a defined reference point, given in aircraft coordinates. It is further recommended that all subsystem models (aerodynamic, propulsive, alighting gear) provide total forces and moments about this reference point and leave the transfer of moments to the center of mass to the equations of motion.

6.5.3. Decomposition of flight dynamics subsystems

It is recommended that a vehicle's flight dynamics reactions be modeled, at least at the highest level, as aerodynamic, propulsive, and landing/arresting/launch gear models. This is common practice in most aircraft simulation environments familiar to the authors.

6.5.4. Date format in DAVE-ML

The recommended way of representing dates in DAVE-ML documentation, especially date attributes and creation date elements, is numerically in the order YYYY-MM-DD. Thus, July 15, 2003 is given as 2003-07-15. This formatting convention conforms to the ISO-8601 standard for representing dates. [ISO8601]

6.5.5. Common sign convention notation

A convention for indicating positive sign conventions for measurements is included in Annex A of ANSI/AIAA S-119-2011 Flight Dynamics Model Exchange Standard [AIAA11]. These acronyms, if applicable, should be used whenever possible to enhance communications.

6.5.6. Units-of-measure abbreviation

Each variable definition includes a mandatory `units` attribute. This attribute gives the units-of-measure for the signal represented by the variable and either 'nd' (for non-dimensional) or blank

if the signal is a dimensionless quantity or flag. In addition, the use of 'fract' to indicate a fraction (0-1) or 'pct' to indicate a percentage (0-100 or more) is encouraged.

Informally, this attribute can take on any reasonable abbreviation for a set of units that might be understandable by the intended audience, in either set of units (English or ISO). For greater re-usability, it is recommended that the set of measurements listed in the *AIAA Flight Dynamics Model Exchange Standard* [AIAA11] (of which this document is a part) be used. The Standard recommends how to encode powers of units (f_s2 for ft/sec^2, for example).

6.5.7. Internal identifiers

Identifiers are used throughout DAVE-ML to connect signals, functions, modification records and reference documents with each other, e.g., `utID`, `gtID`, `bpID`, `refID`, etc. These identifiers are character strings that must comply with the XML specification for Names [W3C-XML]. They must start with a character or underbar, may not start with "xml" or "XML," and may not contain colons, among other restrictions. In addition, the identifiers must be unique within a single DAVE-ML file. See the XML [W3C-XML] (p. 127) for more information regarding valid XML Names.

6.5.8. DAVE-ML Namespace

The XML standard allows for namespace domains, in which element names belong to either the empty (null) namespace or to a namespace that belongs to a particular XML grammar. Namespaces are identified by a uniform resource identifier (URI) that is fashioned, similar to a URL, in order that uniqueness is guaranteed.

The DAVE-ML namespace should be defined in the top-level element as follows:

```
<DAVEfunc xmlns="http://daveml.org/2010/DAVEML">
```

This will allow DAVE-ML models to be embedded in other XML documents without confusion. Note that this general URI is not a URL; HTTP queries at that address may not lead to any useful information.

The `reference` element can include two elements that belong to the World-Wide Web Consortium (W3C)'s XLINK protocol [W3C-XLINK]; they are defined with an `xlink:` prefix which actually refers to the namespace uniquely defined with the `http://www.w3c.org/1999/xlink` URI. If external links to documents will be included in a DAVE-ML document, the top-level element (currently `reference`) *must* include a namespace declaration (which looks like, but technically is not, an attribute):

```
<reference xmlns:xlink="http://www.w3.org/1999/xlink">
```

Similarly, the MathML-2 elements are normally defined in a MathML namespace, so any calculation defined using MathML-2 notation should be conducted inside the MathML namespace:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
```

6.6. Planned major elements

Additional major elements may be defined to support the goal of rapid exchange of simulation models, including

- Support for vector and matrix variables and math operations.
- Subsystem models, to support hierarchical decomposition and problem abstraction.
- State variables, both discrete and continuous, to support dynamic models.
- Dynamic (time-history) data file format, to allow for validation check-cases for dynamic models

7. Further information

Further information, background, and the latest DTD and example models of some aircraft data packages can be found at the DAVE-ML web site: http://daveml.org

8. Element references and descriptions

This section lists the XML tags, or elements, that make up the DAVE-ML grammar. They are listed alphabetically in two sub-sections; the first is a short description of each element; the second sub-section gives details on the element, attributes, and sub-elements.

8.1. Alphabetical list of elements

`address`	Street address or other contact information of an author [Deprecated.]
`author`	Gives name and contact information for originating party of the associated data
`bounds`	Describes limits or standard deviations of statistical uncertainties
`bpRef`	Reference to a breakpoint definition
`bpVals`	String of white space- or comma-separated values of breakpoints
`breakpointDef`	Defines breakpoint sets to be used in model
`breakpointRefs`	A list of breakpoint reference (bpRefs)
`calculation`	Used to delimit a MathML v2 calculation
`checkData`	Gives verification data for encoded model
`checkInputs`	Lists input values for check case
`checkOutputs`	Lists output values for check case
`confidenceBound`	Defines the confidence in a function [Deprecated]
`contactInfo`	Provides multiple contact information associated with an author or agency
`correlatesWith`	Identifies other functions or variables whose uncertainty correlates with our random value
`correlation`	Indicates the linear correlation of this function's or variable's randomness with a previously computed random variable
`creationDate`	Gives date of creation of entity
`dataPoint`	Defines each point of an ungridded table
`dataTable`	Lists the values of a table of function or uncertainty data

`DAVEfunc`	Root level element
`dependentVarPts`	Defines output breakpoint values
`dependentVarRef`	Identifies the signal to be associated with the output of a function
`description`	Verbal description of an entity
`documentRef`	Reference to an external document
`extraDocRef`	Allows multiple documents to be associated with a single modification event
`fileCreationDate`	Gives date of creation of entity [Deprecated]
`fileHeader`	States source and purpose of file
`fileVersion`	Indicates the version of the document
`function`	Defines a function by combining independent variables, breakpoints, and tables
`functionCreationDate`	Date of creation of a function table [Deprecated]
`functionDefn`	Defines a function by associating a table with other information
`griddedTable`	Definition of a gridded table; associates breakpoint data with table data [Deprecated]
`griddedTableDef`	Defines an orthogonally gridded table of data points
`griddedTableRef`	Reference to a gridded table definition
`independentVarPts`	Simple definition of independent breakpoints
`independentVarRef`	References a predefined signal as an input to a function
`internalValues`	A dump of internal model values for debugging check-cases
`isControl`	Flag to identify a model control parameter
`isDisturbance`	Flag to identify a model disturbance input
`isInput`	Flag to identify a model input variable
`isOutput`	Flag to identify non-obvious output signals from model
`isState`	Flag to identify a state variable within a dynamic model
`isStateDeriv`	Flag to identify a state derivative within a dynamic model

`isStdAIAA`	Flag to identify standard AIAA simulation variable
`math`	Denotes a MathML expression
`modificationRecord`	To associate a reference single letter with a modification event
`modificationRef`	Reference to associated modification information
`normalPDF`	Defines a normal (Gaussian) probability density function
`provenance`	Describes origin or history of the associated information
`provenanceRef`	References a previously defined data provenance description
`reference`	Describes an external document
`signal`	Documents an internal DAVE-ML signal (value, units, etc.)
`signalID`	Gives the internal identifier of a varDef [Deprecated]
`signalName`	Gives the external name of an input or output signal
`signalUnits`	Gives the unit-of-measure of an input or output variable
`signalValue`	Gives the value of a check-case signal/variable
`staticShot`	Used to check the validity of the model once instantiated by the receiving facility or tool.
`tol`	Specifies the tolerance of value matching for model verification
`uncertainty`	Describes statistical uncertainty bounds and any correlations for a parameter or function table.
`ungriddedTable`	Definition of an ungridded set of function data [Deprecated]
`ungriddedTableDef`	Defines a table of data, each with independent coordinates
`ungriddedTableRef`	Reference to an ungridded table
`uniformPDF`	Defines a uniform (constant) probability density function
`variableDef`	Defines signals used in DAVE-ML model
`variableRef`	Reference to a variable definition
`varID`	Gives the internal identifier of a varDef

8.2. Element descriptions

This section lists each element in detail, giving the name, content model, attributes, possible parent elements, allowable children elements, and any future plans for the element (such as deprecation).

`address`

address — Street address or other contact information of an author [Deprecated.]

Content model

```
address :
     (#PCDATA)
```

Attributes

NONE

Possible parents

`author`

Allowable children

NONE

Future plans for this element

This element has been subsumed by the contactInfo element below.

`author`

author — Gives name and contact information for originating party of the associated data

Content model

```
author :  name,  org,  [xns],  [email]
     (address* | contactInfo*)
```

Attributes

`name`	The name of the author or last modifier of the associated element's data.
`org`	The author's organization.
`xns`	(optional) (deprecated) The eXtensible Name Service identifier for the author.
`email`	(optional) (deprecated) The e-mail address for the primary author.

Description

author includes alternate means of identifying author using XNS or normal e-mail/address. The address sub-element is to be replaced with the more complete contactInfo sub-element.

Possible parents

```
fileHeader
modificationRecord
provenance
```

Allowable children

```
address
contactInfo
```

Future plans for this element

Both the xns and email attributes are deprecated and will be removed. XNS was a proposed Internet technology (eXtensible Name Service) to reduce spam that didn't catch on. It is replaced with the 'iname' sub-element as a single means to identify an individual or corporation in lieu of typical (and quickly dated) e-mail, phone, or address information. The address element itself is deprecated and should be replaced with the contactInfo element

bounds

bounds — Describes limits or standard deviations of statistical uncertainties

Content model

```
bounds :
     (#PCDATA | dataTable | variableDef | variableRef)*
```

Attributes

NONE

Description

This element contains some description of the statistical limits to the values the citing parameter element might take on. This can be in the form of a scalar value, a private dataTable, or a variableRef. In the more common instance, this element will either be a scalar constant value or a simple table whose dimensions must match the parent nominal function table and whose independent variables are identical to the nominal table. It is also possible that this limit be determined by an independent variable, either previously defined or defined in-line with this element. It does not make sense to have a dataTable cited if this bounds element is associated with anything other than an identically shaped function table.

Possible parents

```
normalPDF
uniformPDF
```

Allowable children

```
dataTable
variableDef
variableRef
```

bpRef

bpRef — Reference to a breakpoint definition

Content model

```
bpRef :  bpID
     EMPTY
```

Attributes

`bpID` The internal identifier for a breakpoint set definition.

Description

The bpRef element provides references to a previously-defined breakpoint set so breakpoints can be defined separately from, and reused by, several data tables.

Possible parents

`breakpointRefs`

Allowable children

NONE

`bpVals`

bpVals — String of white space- or comma-separated values of breakpoints

Content model

```
bpVals :
     (#PCDATA)
```

Attributes

NONE

Description

bpVals is a set of breakpoints (i.e., a set of independent variable values associated with one dimension of a gridded table of data). An example would be the Mach or angle-of-attack values that define the coordinates of each data point in a 2D coefficient value table.

Possible parents

`breakpointDef`

Allowable children

NONE

`breakpointDef`

breakpointDef — Defines breakpoint sets to be used in model

Content model

```
breakpointDef :  [name],  bpID,  [units]
     description?
     bpVals
```

Attributes

`name`	(optional) The name of the breakpoint set.
`bpID`	The internal identifier for the breakpoint set.
`units`	(optional) The units of measure for the breakpoint set.

Description

A breakpointDef lists gridded table breakpoints. Since these are separate from function data they may be reused.

Possible parents

`DAVEfunc`

Allowable children

```
description
bpVals
```

`breakpointRefs`

breakpointRefs — A list of breakpoint reference (bpRefs)

Content model

```
breakpointRefs :
     bpRef+
```

Attributes

NONE

Description

The breakpointRefs elements tie the independent variable names for the function to specific breakpoint values defined earlier.

Possible parents

`griddedTableDef`
`griddedTable`

Allowable children

`bpRef`

`calculation`

calculation — Used to delimit a MathML v2 calculation

Content model

```
calculation :
     math
```

Attributes

NONE

Description

The calculation element wraps around MathML 2 content markup describing how the signal is calculated. The calculation may include both constants and variables; other variables are included by using their varID string in a MathML content identifier (ci) element. The calculation element must have one and only one subelement, and the subelement must be a 'math' element containing 'content' markup.

Possible parents

`variableDef`

Allowable children

`math`

`checkData`

checkData — Gives verification data for encoded model

Content model

```
checkData :
     (provenance | provenanceRef)?
     staticShot+
```

Attributes

NONE

Description

This top-level element is the place-holder for verification data of various forms for the encoded model. It will include static check cases, trim shots, and dynamic check case information. The provenance sub-element is now deprecated and has been moved to individual staticShots; it is allowed here for backwards compatibility.

Possible parents

`DAVEfunc`

Allowable children

```
provenance
provenanceRef
staticShot
```

`checkInputs`

checkInputs — Lists input values for check case

Content model

```
checkInputs :
     signal+
```

Attributes

NONE

Description

Specifies the contents of the input vector for the given check case.

Possible parents

`staticShot`

Allowable children

`signal`

`checkOutputs`

checkOutputs — Lists output values for check case

Content model

```
checkOutputs :
     signal+
```

Attributes

NONE

Description

Specifies the contents of the output vector for the given check case.

Possible parents

`staticShot`

Allowable children

`signal`

`confidenceBound`

confidenceBound — Defines the confidence in a function [Deprecated]

Content model

```
confidenceBound :  value
     EMPTY
```

Attributes

`value` Percent confidence (like 95%) in the function.

Description

The confidenceBound element is used to declare the confidence interval associated with the data table. This is a place-holder and will be removed in a future version of DAVE-ML.

Possible parents

`griddedTable`
`ungriddedTable`

Allowable children

NONE

Future plans for this element

Deprecated. Used only in deprecated [un]griddedTable elements. Use uncertainty element instead.

`contactInfo`

contactInfo — Provides multiple contact information associated with an author or agency

Content model

```
contactInfo :  [contactInfoType],  [contactLocation]
     (#PCDATA)
```

Attributes

`contactInfoType` (optional) Indicates type of information being conveyed (enumerated).

- `address`
- `phone`
- `fax`
- `email`
- `iname`
- `web`

`contactLocation` (optional) Indicates which location is identified. Default is professional (enumerated).

- `professional`
- `personal`
- `mobile`

Description

Used to provide contact information about an author. Use contactInfoType to differentiate what information is being conveyed and contactLocation to denote location of the address.

Possible parents

`author`

Allowable children

NONE

`correlatesWith`

correlatesWith — Identifies other functions or variables whose uncertainty correlates with our random value

Content model

```
correlatesWith :  varID
     EMPTY
```

Attributes

`varID` Identifies the variable or function output that will depend on this function's or variable's randomness.

Description

When present, this element indicates the parent function or variable is correlated with the referenced other function or variable in a linear sense. This alerts the application that the random number used to calculate this function's or variable's immediate value will be used to calculate another function's or variable's value.

Possible parents

`normalPDF`

Allowable children

NONE

`correlation`

correlation — Indicates the linear correlation of this function's or variable's randomness with a previously computed random variable

Content model

```
correlation :  varID,  corrCoef
     EMPTY
```

Attributes

`varID`	Identifies the variable or function output that helps determine the value of this random variable or function.
`corrCoef`	Indicates the amount of correlation between this variable and the referenced variable. The value should be between #1 and +1; 0 indicates no correlation, +1 indicates perfect correlation and #1 indicates inverse (negative) correlation.

Description

When present, this element indicates the parent function or variable is correlated with the referenced other function or variable in a linear sense and gives the correlation coefficient for determining this function's random value based upon the correlating function(s)'s random value.

Possible parents

`normalPDF`

Allowable children

NONE

`creationDate`

creationDate — Gives date of creation of entity

Content model

```
creationDate :  date
     EMPTY
```

Attributes

`date` The date of creation of the entity, in ISO 8601 (YYYY-MM-DD) format.

Description

creationDate is simply a string with a date in it. We follow ISO 8601 and use dates like "2004-01-02" to refer to January 2, 2004.

Possible parents

`fileHeader`
`provenance`

Allowable children

NONE

`dataPoint`

dataPoint — Defines each point of an ungridded table

Content model

```
dataPoint :  [modID]
     (#PCDATA)
```

Attributes

`modID` (optional) The internal identifier for a modification record.

Description

The dataPoint element is used by ungridded tables to list the values of independent variables that are associated with each value of dependent variable. For example: <dataPoint> 0.1, -4.0, 0.2 <!- Mach, alpha, CL -> </dataPoint> <dataPoint> 0.1, 0.0, 0.6 <!- Mach, alpha CL -> </dataPoint> Each data point may have associated with it a modification tag to document the genesis of that particular point. No requirement on ordering of independent variables is implied. Since this is an ungridded table, the interpreting application is required to handle what may be unsorted data.

Possible parents

`ungriddedTableDef`
`ungriddedTable`

Allowable children

NONE

`dataTable`

dataTable — Lists the values of a table of function or uncertainty data

Content model

```
dataTable :
     (#PCDATA)
```

Attributes

NONE

Description

The dataTable element is used by gridded tables where the indep. variable values are implied by breakpoint sets. Thus, the data embedded between the dataTable element tags is expected to be sorted ASCII values of the gridded table, wherein the last independent variable listed in the function header varies most rapidly. The table data point values are specified as comma- or white space-separated values in conventional floating-point notation (0.93638E-06) in a single long sequence as if the table had been unraveled with the last-specified dimension changing most rapidly. Line breaks are to be ignored. Comments may be embedded in the table to promote [human] readability, with appropriate escaping characters. A dataTable element can also be used in an uncertainty element to provide duplicate uncertainty bound values.

Possible parents

`griddedTableDef`
`griddedTable`
`bounds`

Allowable children

NONE

DAVEfunc

DAVEfunc — Root level element

Content model

```
DAVEfunc :  xmlns
     fileHeader
     variableDef+
     breakpointDef*
     griddedTableDef*
     ungriddedTableDef*
     function*
     checkData?
```

Attributes

`xmlns` This attribute specifies that the default namespace for un-prefixed elements is this DTD.

Description

Root element is DAVEfunc, composed of a file header element followed by one or more variable definitions and zero or more breakpoint definitions, gridded or ungridded table definitions, and function elements.

Possible parents

NONE - ROOT ELEMENT

Allowable children

```
fileHeader
variableDef
breakpointDef
griddedTableDef
ungriddedTableDef
function
checkData
```

`dependentVarPts`

dependentVarPts — Defines output breakpoint values

Content model

```
dependentVarPts :  varID,  [name],  [units],  [sign]
     (#PCDATA)
```

Attributes

`varID`	The internal identifier of the output signal this table should drive.
`name`	(optional) The name of the function's dependent variable output signal.
`units`	(optional) The units of measure for the dependent variable.
`sign`	(optional) The sign convention for the dependent variable.

Description

A dependentVarPts element is a simple comma- or white space-delimited list of function values and contains a mandatory varID as well as optional name, units, and sign convention attributes. Data points are arranged as single vector with last-specified breakpoint values changing most frequently. Note that the number of dependent values must equal the product of the number of independent values for this simple, gridded, realization. This element is used for simple functions that do not share breakpoint or table values with other functions.

Possible parents

`function`

Allowable children

NONE

`dependentVarRef`

dependentVarRef — Identifies the signal to be associated with the output of a function

Content model

```
dependentVarRef :  varID
     EMPTY
```

Attributes

`varID` The internal identifier for the output signal.

Description

A dependentVarRef ties the output of a function to a signal name defined previously in a variable definition.

Possible parents

`function`

Allowable children

NONE

`description`

description — Verbal description of an entity

Content model

```
description :
     (#PCDATA)
```

Attributes

NONE

Description

The description element is a textual description of an entity. The full UNICODE character set is supported by XML but may not be available in all processing applications.

Possible parents

```
fileHeader
variableDef
breakpointDef
griddedTableDef
ungriddedTableDef
function
reference
modificationRecord
provenance
staticShot
```

Allowable children

NONE

`documentRef`

documentRef — Reference to an external document

Content model

```
documentRef :  [docID],  refID
     EMPTY
```

Attributes

`docID` (optional) (deprecated) The internal identifier of a reference definition element.

`refID` The internal identifier of a reference definition element.

Possible parents

`provenance`

Allowable children

NONE

Future plans for this element

The 'docID' attribute is deprecated; it has been renamed 'refID' to match its use in the 'reference' element. This attribute will be removed in a future version of DAVE-ML.

`extraDocRef`

extraDocRef — Allows multiple documents to be associated with a single modification event

Content model

```
extraDocRef :  refID
     EMPTY
```

Attributes

`refID` The internal identifier of a reference definition element.

Description

A single modification event may have more than one documented reference. This element can be used in place of the refID attribute in a modificationRecord to record more than one refIDs, pointing to the referenced document.

Possible parents

`modificationRecord`

Allowable children

NONE

`fileCreationDate`

fileCreationDate — Gives date of creation of entity [Deprecated]

Content model

```
fileCreationDate :  date
     EMPTY
```

Attributes

`date` The date of the file, in ISO 8601 (YYYY-MM-DD) format.

Description

fileCreationDate is simply a string with a date in it. We follow ISO 8601 and use dates like "2004-01-02" to refer to January 2, 2004. Its use is now deprecated in favor of the simpler creationDate.

Possible parents

`fileHeader`

Allowable children

NONE

Future plans for this element

The fileCreationDate and functionCreationDate have been replaced with a new creationDate multipurpose element.

`fileHeader`

fileHeader — States source and purpose of file

Content model

```
fileHeader :  [name]
     author+
     (creationDate | fileCreationDate)
     fileVersion?
     description?
     reference*
     modificationRecord*
     provenance*
```

Attributes

`name` (optional) The name of the file.

Description

The header element requires at least one author and a creation date; optional content includes version indicator, description, references, and modification records.

Possible parents

`DAVEfunc`

Allowable children

`author`
`creationDate`
`fileCreationDate`
`fileVersion`
`description`
`reference`
`modificationRecord`
`provenance`

`fileVersion`

fileVersion — Indicates the version of the document

Content model

```
fileVersion :
     (#PCDATA)
```

Attributes

NONE

Description

This is a string describing, in some arbitrary text, the version identifier for this function description.

Possible parents

`fileHeader`

Allowable children

NONE

`function`

function — Defines a function by combining independent variables, breakpoints, and tables

Content model

```
function :  name
     description?
     (provenance | provenanceRef)?
     (
          independentVarPts+
          dependentVarPts
     )
     OR
     (
          independentVarRef+
          dependentVarRef
          functionDefn
     )
```

Attributes

`name` The name of this function.

Description

Each function has optional description, optional provenance, and either a simple input/output table values or references to more complete (possible multiple) input, output, and function data elements.

Possible parents

`DAVEfunc`

Allowable children

`description`
`provenance`
`provenanceRef`
`independentVarPts`
`dependentVarPts`
`independentVarRef`
`dependentVarRef`
`functionDefn`

`functionCreationDate`

functionCreationDate — Date of creation of a function table [Deprecated]

Content model

```
functionCreationDate :  date
     EMPTY
```

Attributes

`date` The creation date of the function, in ISO 8601 (YYYY-MM-DD) format.

Description

functionCreationDate is simply a string with a date in it. We follow ISO 8601 and use dates like "2004-01-02" to refer to January 2, 2004. Its use is now deprecated in favor of the simpler creationDate.

Possible parents

`provenance`

Allowable children

NONE

Future plans for this element

The fileCreationDate and functionCreationDate have been replaced with a new creationDate multipurpose element.

`functionDefn`

functionDefn — Defines a function by associating a table with other information

Content model

```
functionDefn :  [name]
     (griddedTableRef | griddedTableDef | griddedTable | ungriddedTableRef
 | ungriddedTableDef | ungriddedTable)
```

Attributes

`name` (optional) The name of this function definition.

Description

A functionDefn defines how function is represented in one of two possible ways: gridded (implies breakpoints) or ungridded (with explicit independent values for each point).

Possible parents

`function`

Allowable children

```
griddedTableRef
griddedTableDef
griddedTable
ungriddedTableRef
ungriddedTableDef
ungriddedTable
```

`griddedTable`

griddedTable — Definition of a gridded table; associates breakpoint data with table data [Deprecated]

Content model

```
griddedTable :  [name]
     breakpointRefs
     confidenceBound?
     dataTable
```

Attributes

`name` (optional) The name of the gridded table being defined.

Possible parents

`functionDefn`

Allowable children

`breakpointRefs`
`confidenceBound`
`dataTable`

Future plans for this element

Deprecated. Use griddedTableDef instead.

griddedTableDef

griddedTableDef — Defines an orthogonally gridded table of data points

Content model

```
griddedTableDef :  [name],  gtID,  [units]
     description?
     (provenance | provenanceRef)?
     breakpointRefs
     uncertainty?
     dataTable
```

Attributes

name	(optional) The name of the gridded table.
gtID	An internal identifier for the table. As of DAVE-ML 2.0, it is required for all tables (previously was required only if the table was to be be reused by another function or was defined outside of a function.
units	(optional) Units of measure for the table values.

Description

A griddedTableDef contains points arranged in an orthogonal (but multi-dimensional) array, where the independent variables are defined by separate breakpoint vectors. This table definition may be specified separately from the actual function declaration.

Possible parents

DAVEfunc
functionDefn

Allowable children

description
provenance
provenanceRef
breakpointRefs
uncertainty
dataTable

griddedTableRef

griddedTableRef — Reference to a gridded table definition

Content model

```
griddedTableRef :  gtID
     EMPTY
```

Attributes

`gtID` The internal identifier of a gridded table definition.

Possible parents

`functionDefn`

Allowable children

NONE

`independentVarPts`

independentVarPts — Simple definition of independent breakpoints

Content model

```
independentVarPts : varID, [name], [units], [sign], [extrapolate], [interpolate]
     (#PCDATA)
```

Attributes

`varID`	The internal identifier of the input signal corresponding to this independent variable.
`name`	(optional) The name of the function's independent variable input signal.
`units`	(optional) The units of measure for the independent variable.
`sign`	(optional) The sign convention for the independent variable.
`extrapolate`	(optional) Extrapolation flags for IV out-of-bounds (default is neither) (enumerated). • `neither` • `min` • `max` • `both`
`interpolate`	(optional) Interpolation flags for independent variable (default is linear) (enumerated). • `discrete` • `floor` • `ceiling` • `linear` • `quadraticSpline` • `cubicSpline`

Description

An independentVarPts element is a simple white space- or comma-separated list of breakpoints and contains a mandatory varID identifier as well as optional name, units, and sign convention

attributes. An optional extrapolate attribute describes how to extrapolate the output value when the input value exceeds specified values (default is 'neither,' meaning the value of the table is held constant at the nearest defined value). An optional interpolate attribute indicates how to perform the interpolation within the table (supporting discrete, linear, cubic or quadratic splines). There are three different discrete options: 'discrete' means nearest-neighbor, with an exact mid-point value being rounded in the positive direction; 'ceiling' means the function takes on the value associated with the next (numerically) higher independent breakpoint as soon as the original value is exceeded; and 'floor' means the function holds the value associated with each breakpoint until the next (numerically) higher breakpoint value is reached by the independent argument. The default interpolation attribute value is 'linear.' This element is used for simple functions that do not share breakpoint or table values with other functions.

Possible parents

`function`

Allowable children

NONE

`independentVarRef`

independentVarRef — References a predefined signal as an input to a function

Content model

```
independentVarRef :  varID,  [min],  [max],  [extrapolate],  [interpolate]
     EMPTY
```

Attributes

`varID`	The internal identifier for the input signal.
`min`	(optional) The allowable lower limit for the input signal.
`max`	(optional) The allowable upper limit for the input signal.
`extrapolate`	(optional) Extrapolation flags for IV out-of-bounds (default is neither) (enumerated). • `neither` • `min` • `max` • `both`
`interpolate`	(optional) Interpolation flags for independent variable (default is linear) (enumerated). • `discrete` • `floor` • `ceiling` • `linear` • `quadraticSpline` • `cubicSpline`

Description

An independentVarRef more fully describes the input mapping of the function by pointing to a separate breakpoint definition element. An optional extrapolate attribute describes how to extrapolate the output value when the input value exceeds specified values (default is 'neither,'

meaning the value of the table is held constant at the nearest defined value). An optional interpolate attribute indicates how to perform the interpolation within the table (supporting discrete, linear, cubic or quadratic splines). There are three different discrete options: 'discrete' means nearest-neighbor, with an exact mid-point value being rounded in the positive direction; 'floor' means the function takes on the value associated with the next (numerically) higher independent breakpoint as soon as original value is exceeded; and 'ceiling' means the function holds the value associated with each breakpoint until the next (numerically) higher breakpoint value is reached by the independent argument. The default interpolation attribute value is 'linear.' Note also that the option min and max attributes serve to limit the numeric range of the independent (function input) variable; these attributes are independent of and serve as additional range contraints with respect to any minValue or maxValue attribute associated with the independent variable itself. This element allows reuse of common breakpoint values for many tables but with possible differences in interpolation or extrapolation for each use.

Possible parents

`function`

Allowable children

NONE

`internalValues`

internalValues — A dump of internal model values for debugging check-cases

Content model

```
internalValues :
     signal+
```

Attributes

NONE

Description

Provides a set of all internal variable values to assist in debugging recalcitrant implementations of DAVE-ML import tools.

Possible parents

`staticShot`

Allowable children

`signal`

`isControl`

isControl — Flag to identify a model control parameter

Content model

```
isControl :
     EMPTY
```

Attributes

NONE

Description

The presence of an isControl element indicates that this signal is a simulation control parameter used to vary the operation of the model, e.g. the time step size. Such parameters should be ignored when performing linear model extraction (for example) and should not significantly modify the dynamic behavior of the model.

Possible parents

`variableDef`

Allowable children

NONE

`isDisturbance`

isDisturbance — Flag to identify a model disturbance input

Content model

```
isDisturbance :
     EMPTY
```

Attributes

NONE

Description

The presence of an isDisturbance element indicates that this signal is an external disturbance input to the model and can be ignored when performing linear model extraction (for example). Such parameters should not significantly modify the nominal dynamic behavior of the model.

Possible parents

`variableDef`

Allowable children

NONE

`isInput`

isInput — Flag to identify a model input variable

Content model

```
isInput :
     EMPTY
```

Attributes

NONE

Description

The presence of an isInput element indicates that this variable is an input signal to the model.

Possible parents

`variableDef`

Allowable children

NONE

`isOutput`

isOutput — Flag to identify non-obvious output signals from model

Content model

```
isOutput :
     EMPTY
```

Attributes

NONE

Description

The presence of the isOutput element indicates that this variable should be forced to be an output, even if it is used internally as an input elsewhere. Otherwise, the processing program may assume a signal defined with a calculation and used subsequently in the model is only an internal signal.

Possible parents

`variableDef`

Allowable children

NONE

`isState`

isState — Flag to identify a state variable within a dynamic model

Content model

```
isState :
     EMPTY
```

Attributes

NONE

Description

The presence of an isState element indicates that this variable is one of possibly multiple state variables in a dynamic model; this tells the processing entity that this is the output of an integrator (for continuous models) or a discretely updated state (for discrete models).

Possible parents

`variableDef`

Allowable children

NONE

`isStateDeriv`

isStateDeriv — Flag to identify a state derivative within a dynamic model

Content model

```
isStateDeriv :
     EMPTY
```

Attributes

NONE

Description

The presence of an isStateDeriv element indicates that this variable is one of possibly several state derivative variables in a dynamic model; this tells the processing entity that this is the output of an integrator (for continuous models only).

Possible parents

`variableDef`

Allowable children

NONE

`isStdAIAA`

isStdAIAA — Flag to identify standard AIAA simulation variable

Content model

```
isStdAIAA :
     EMPTY
```

Attributes

NONE

Description

The presence of an isStdAIAA element indicates that this variable is one of the standard AIAA variable names which should be recognizable exterior to this module (e.g. AngleOfAttack_deg). This flag should assist importing tools in determining when an input or output should match a facility-provided signal name without requiring further information.

Possible parents

`variableDef`

Allowable children

NONE

`math`

math — Denotes a MathML expression

Content model

```
math :  xmlns
```

Attributes

`xmlns`	The value of this attribute must be "http://www.w3.org/1998/Math/MathML". If omitted, it should be provided by the parser since it is specified in the DTD.

Description

The math element denotes MathML 2 content markup and should include the attribute 'xmlns="http://www.w3.org/1998/Math/MathML"'

Possible parents

`calculation`

Allowable children

NONE

`modificationRecord`

modificationRecord — To associate a reference single letter with a modification event

Content model

```
modificationRecord :  modID,  date,  [refID]
     author+
     description?
     extraDocRef*
```

Attributes

`modID` A single letter used to identify all modified data associated with this modification record.

`date` The date of the modification, in ISO 8601 (YYYY-MM-DD) format.

`refID` (optional) A reference to a predefined document that describes the reason for this modification.

Description

A modificationRecord associates a single letter (such as modification "A") with modification author(s), address, and any optional external reference documents, in keeping with the AIAA draft standard.

Possible parents

`fileHeader`

Allowable children

```
author
description
extraDocRef
```

`modificationRef`

modificationRef — Reference to associated modification information

Content model

```
modificationRef :  modID
     EMPTY
```

Attributes

`modID` The internal identifier of a modification definition.

Possible parents

`provenance`

Allowable children

NONE

normalPDF

normalPDF — Defines a normal (Gaussian) probability density function

Content model

```
normalPDF :  numSigmas
     bounds
     correlatesWith*
     correlation*
```

Attributes

numSigmas	Indicates how many standard deviations is represented by the uncertainty values given later. Integer value > 0.

Description

In a normally distributed random variable, a symmetrical distribution of given standard deviation is assumed about the nominal value (which is given elsewhere in the parent element). The correlatesWith sub-element references other functions or variables that have a linear correlation to the current parameter or function. The correlation sub-element specifies the correlation coefficient and references the other function or variable whose random value helps determine the value of this parameter.

Possible parents

uncertainty

Allowable children

bounds
correlatesWith
correlation

provenance

provenance — Describes origin or history of the associated information

Content model

```
provenance :  [provID]
     author+
     (creationDate | functionCreationDate)
     documentRef*
     modificationRef*
     description?
```

Attributes

`provID` (optional) This attribute allows the provenance information defined here to be cited elsewhere.

Description

The provenance element describes the history or source of the model data and includes author, date, and zero or more references to documents and modification records.

Possible parents

`fileHeader`
`variableDef`
`griddedTableDef`
`ungriddedTableDef`
`function`
`checkData`
`staticShot`

Allowable children

`author`
`creationDate`
`functionCreationDate`
`documentRef`
`modificationRef`
`description`

`provenanceRef`

provenanceRef — References a previously defined data provenance description

Content model

```
provenanceRef :  provID
     EMPTY
```

Attributes

`provID` The internal identifier for a previously defined provenance.

Description

When the provenance of a set of several data is identical, the first provenance element should be given a provID and referenced by later provenanceRef elements.

Possible parents

`variableDef`
`griddedTableDef`
`ungriddedTableDef`
`function`
`checkData`
`staticShot`

Allowable children

NONE

`reference`

reference — Describes an external document

Content model

```
reference :  xmlns:xlink,  xlink:type,  refID,  author,  title,  [classification],
 [accession],  date,  [xlink:href]
     description?
```

Attributes

`xmlns:xlink`	The value of this attribute must be "http://www.w3.org/1999/xlink". If omitted, it should be provided by the parser since it is specified in the DTD.
`xlink:type`	The value of this attribute must be "simple". If omitted, it should be provided by the parser since it is specified in the DTD.
`refID`	An internal identifier for this reference definition.
`author`	The name of the author of the reference.
`title`	The title of the referenced document.
`classification`	(optional) The security classification of the document.
`accession`	(optional) The accession number (ISBN or organization report number) of the document.
`date`	The date of the document, in ISO 8601 (YYYY-MM-DD) format.
`xlink:href`	(optional) A URL to an on-line copy of the referenced document.

Description

This element gives identifying (citation) information to an external, possibly on-line, reference document, including a user-specified author, title, classification, accession number, date and URL.

Possible parents

`fileHeader`

Allowable children

`description`

`signal`

signal — Documents an internal DAVE-ML signal (value, units, etc.)

Content model

```
signal :
     (
          signalName
          signalUnits
     )
     OR
     (
          (varID | signalID)
     )
     signalValue
     tol?
```

Attributes

NONE

Description

This element is used to document the name, ID, value, tolerance, and units of measure for check-cases. When used with checkInputs or checkOutputs, the signalName sub-element must be present (since check cases are viewed from "outside" the model); when used in an internalValues element, the varID sub-element should be used to identify the signal by its model-unique internal reference. When used in a checkOutputs vector, the tol element must be present. Tolerance is specified as a maximum absolute difference between the expected and actual value. The signalID sub-element is now deprecated in favor of the more consistent varID.

Possible parents

```
checkInputs
internalValues
checkOutputs
```

Allowable children

```
signalName
signalUnits
varID
signalID
signalValue
tol
```

`signalID`

signalID — Gives the internal identifier of a varDef [Deprecated]

Content model

```
signalID :
     (#PCDATA)
```

Attributes

NONE

Description

Used to specify the input or output varID. Now deprecated; reuse of varID is best practice.

Possible parents

`signal`

Allowable children

NONE

Future plans for this element

The signalID element has been deprecated with 2.0 in favor of the more consistent varID element and will be unsupported in a future release of this specification.

`signalName`

signalName — Gives the external name of an input or output signal

Content model

```
signalName :
     (#PCDATA)
```

Attributes

NONE

Description

Used inside a checkCase element to specify the input or output variable name

Possible parents

`signal`

Allowable children

NONE

`signalUnits`

signalUnits — Gives the unit-of-measure of an input or output variable

Content model

```
signalUnits :
     (#PCDATA)
```

Attributes

NONE

Description

Used inside a checkCase element to specify the units-of-measure for an input or output variable, for verification of proper implementation of a model.

Possible parents

`signal`

Allowable children

NONE

`signalValue`

signalValue — Gives the value of a check-case signal/variable

Content model

```
signalValue :
     (#PCDATA)
```

Attributes

NONE

Description

Used to give the current value of an internal signal or input/output variable, for verification of proper implementation of a model.

Possible parents

`signal`

Allowable children

NONE

staticShot

staticShot — Used to check the validity of the model once instantiated by the receiving facility or tool.

Content model

```
staticShot :  name,  [refID]
     description?
     (provenance | provenanceRef)?
     checkInputs?
     internalValues?
     checkOutputs
```

Attributes

`name`

`refID` (optional) Points to a reference given in the file header.

Description

Contains a description of the inputs and outputs, and possibly internal values, of a DAVE-ML model in a particular instant of time.

Possible parents

`checkData`

Allowable children

`description`
`provenance`
`provenanceRef`
`checkInputs`
`internalValues`
`checkOutputs`

`tol`

tol — Specifies the tolerance of value matching for model verification

Content model

```
tol :
     (#PCDATA)
```

Attributes

NONE

Description

This element specifies the allowable tolerance of error in an output value such that the model can be considered verified. It is assumed all uncertainty is removed in performing the model calculations. Tolerance is specified as a maximum absolute difference between the expected and actual value.

Possible parents

`signal`

Allowable children

NONE

`uncertainty`

uncertainty — Describes statistical uncertainty bounds and any correlations for a parameter or function table.

Content model

```
uncertainty :  effect
     (normalPDF | uniformPDF)
```

Attributes

`effect` Indicates how uncertainty bounds are interpreted. (enumerated).

- `additive`
- `multiplicative`
- `percentage`
- `absolute`

Description

The uncertainty element is used in function and parameter definitions to describe statistical variance in the possible value of that function or parameter value. Only Gaussian (normal) or uniform distributions of continuous random variable distribution functions are supported.

Possible parents

`variableDef`
`griddedTableDef`
`ungriddedTableDef`

Allowable children

`normalPDF`
`uniformPDF`

`ungriddedTable`

ungriddedTable — Definition of an ungridded set of function data [Deprecated]

Content model

```
ungriddedTable :  [name]
     confidenceBound?
     dataPoint+
```

Attributes

`name` (optional) The name of the ungridded table being defined.

Possible parents

`functionDefn`

Allowable children

`confidenceBound`
`dataPoint`

Future plans for this element

Deprecated. Use ungriddedTableDef instead.

`ungriddedTableDef`

ungriddedTableDef — Defines a table of data, each with independent coordinates

Content model

```
ungriddedTableDef :  [name],  utID,  [units]
     description?
     (provenance | provenanceRef)?
     uncertainty?
     dataPoint+
```

Attributes

`name` (optional) The name of the ungridded table.

`utID` An internal identifier for the table. As of DAVE-ML 2.0, it is required for all tables (previously it was required only if the table was to be be reused by another function or was defined outside of a function.

`units` (optional) The units of measure for the table values.

Description

An ungriddedTableDef contains points that are not in an orthogonal grid pattern; thus, the independent variable coordinates are specified for each dependent variable value. This table definition may be specified separately from the actual function declaration.

Possible parents

`DAVEfunc`
`functionDefn`

Allowable children

`description`
`provenance`
`provenanceRef`
`uncertainty`
`dataPoint`

`ungriddedTableRef`

ungriddedTableRef — Reference to an ungridded table

Content model

```
ungriddedTableRef :  utID
     EMPTY
```

Attributes

`utID` The internal identifier of a ungridded table definition.

Possible parents

`functionDefn`

Allowable children

NONE

`uniformPDF`

uniformPDF — Defines a uniform (constant) probability density function

Content model

```
uniformPDF :
     bounds+
```

Attributes

NONE

Description

In a uniformly distributed random variable, the value of the parameter has equal likelihood of assuming any value within the (possibly asymmetric, implied by specifying two) bounds, which must bracket the nominal value (which is given elsewhere in the parent element).

Possible parents

`uncertainty`

Allowable children

`bounds`

`variableDef`

variableDef — Defines signals used in DAVE-ML model

Content model

```
variableDef :  name,  varID,  units,  [axisSystem],  [sign],  [alias],  [symbol],
 [initialValue],  [minValue],  [maxValue]
    description?
    (provenance | provenanceRef)?
    calculation?
    (isInput | isControl | isDisturbance)?
    isState?
    isStateDeriv?
    isOutput?
    isStdAIAA?
    uncertainty?
```

Attributes

`name`	The name of the signal being defined.
`varID`	An internal identifier for the signal.
`units`	The units of the signal.
`axisSystem`	(optional) The axis in which the signal is measured.
`sign`	(optional) The sign convention for the signal, if any.
`alias`	(optional) Possible alias name (facility specific) for the signal.
`symbol`	(optional) UNICODE symbol for the signal.
`initialValue`	(optional) An initial and possibly constant numeric value for the signal.
`minValue`	(optional) A minimum value the variable shall be limited to remain greater than or equal to.
`maxValue`	(optional) A maximum value the variable shall be limited to remain less than or equal to.

Description

variableDef elements provide wiring information (i.e., they identify the input and output signals used by these function blocks). They also provide MathML content markup to indicate any calculation required to arrive at the value of the variable, using other variables as inputs. The variable definition can include statistical information regarding the uncertainty of the values which it might take on, when measured after any calculation is performed. Information about the reason for inclusion or change to this element can be included in an optional provenance sub-element. If either or both min or max attributes are specified, the implementer must limit the

final value of the variable to remain within the specified bound(s). These attributes imply a one- or two-sided limiter function downstream of any specified calculation, input signal, or initial value. Min and max attributes may only be fixed numeric values. To provide variable limits, the modeler should employ the piecewise MathML element in a calculation subelement.

Possible parents

DAVEfunc
bounds

Allowable children

description
provenance
provenanceRef
calculation
isInput
isControl
isDisturbance
isState
isStateDeriv
isOutput
isStdAIAA
uncertainty

`variableRef`

variableRef — Reference to a variable definition

Content model

```
variableRef :  varID
     EMPTY
```

Attributes

`varID` The internal identifier of a previous variable definition.

Possible parents

`bounds`

Allowable children

NONE

`varID`

varID — Gives the internal identifier of a varDef

Content model

```
varID :
     (#PCDATA)
```

Attributes

NONE

Description

Used to specify the input or output varID. Replaces earlier signalID element.

Possible parents

`signal`

Allowable children

NONE

The editors would like to acknowledge the contributions, encouragement and helpful suggestions for steering the evolution of DAVE-ML from the following individuals: Trey Arthur (NASA Langley Research Center), Jon Berndt (Jacobs Sverdrup), Geoff Brian (Australian DSTO), Randy Brumbaugh (Indigo Innovations), Carey Buttrill (NASA Langley Research Center), Giovanni Cignoni (University of Pisa), Bill Cleveland (NASA Ames Research Center), Zack Crues (NASA Johnson Space Center), Rob Falck (NASA Glenn Research Center), Jeremy Furtek (Delphi Research), Peter Grant (UTIAS), David Hasan (L-3 Communications, Houston), Bruce Hildreth (J.F. Taylor, Inc, Lexington Park), Missy Hill (UNISYS), Matt Jessick (L-3 Communications, Houston), Hilary Keating (Fortburn Pty. Ltd.), Dennis Linse (SAIC), Bill Othon (NASA Johnson Space Center), Mike Madden (NASA Langley Research Center), Greg McCarthy (NASA Dryden Flight Research Center), Dana McMinn (NASA Langley Research Center), Dan Murri (NASA Engineering and Safety Center, Hampton), Dan Newman (formerly Ball Aerospace, now Quantitative Aeronautics), John Penn (L-3 Communications, Houston) Riley Rainey (SDS International), Mike Red (NASA Johnson Space Center), Brent York (formerly NAVAIR, now Indra Systems, Inc.), and Curtis Zimmerman (NASA Marshall Space Flight Center).

References

[Acevedo07] : Acevedo, Amanda; Arnold, Jason; Othon, William; and Berndt, Jon : *ANTARES: Spacecraft Simulation for Multiple User Communities and Facilities*. AIAA 2007-6888, presented at the AIAA Modeling and Simulation Technologies Conference, 20 August 2007, Hilton Head, South Carolina.

[AIAA92] : American Institute of Aeronautics and Astronautics : *American National Standard: Recommended Practice for Atmospheric and Space Flight Vehicle Coordinate Systems*. ANSI/ AIAA R-004-1992

[AIAA01] : AIAA Flight Simulation Technical Committee : "Standard Simulation Variable Names [http://daveml.nasa.gov/AIAA_stds/SimParNames_May_2007_v2.pdf]", Draft, May 2007

[AIAA03] : AIAA Modeling and Simulation Technical Committee : "Standards for the Exchange of Simulation Modeling Data [http://daveml.nasa.gov/AIAA_stds/SimDataExchange_Jan2003.pdf]", Preliminary Draft, Jan 2003

[AIAA11] : American Institute of Aeronautics and Astronautics : *American National Standard: Flight Dynamics Model Exchange Standard [https://info.aiaa.org/Standards/AS/DAVE/default.aspx]*. ANSI/AIAA S-119-2011

[Brian05] : Brian, Geoff; Young, Michael; Newman, Daniel; Curtin, Robert; and Keating, Hilary : *The Quest for a Unified Aircraft Dataset Format [http://www.siaa.asn.au/get/2411855949.pdf]* . Presented at the Simulation Industry Association of Australia, 2005.

[Durak06] : Durak, Umut; Oguztuzun, Halit; and Ider, S. Kemal : *An Ontology for Trajectory Simulation* . Proceedings of the 2006 Winter Simulation Conference, 2006.

[Hildreth94] : Hildreth, B. L.: *A Process for the Development of Simulation Standards*. AIAA 94-3430, presented at the AIAA Flight Simulation Technologies Conference, August 1994, Baltimore, Maryland.

[Hildreth98] : Hildreth, Bruce L.: *The Draft AIAA Flight Mechanics Modeling Standard: An Opportunity for Industry Feedback*. AIAA 98-4576, presented at the AIAA Modeling and Simulation Technologies Conference, August 1998, Boston, Massachusetts.

[Hildreth08] : Hildreth, Bruce; Linse, Dennis J.; and Dicola, John : *Pseudo Six Degree of Freedom (6DOF) Models for Higher Fidelity Constructive Simulations*. AIAA 2008-6857, presented at the AIAA Modeling and Simulation Technologies Conference, 18 August 2008, Honolulu, Hawaii.

[Hill07] : Hill, Melissa A.; and Jackson, E. Bruce : *The DaveMLTranslator: An Interface for DAVE-ML Aerodynamic Models [http://http://daveml/papers/2007_AIAA_DaveMLTranslator_paper.pdf]*. AIAA 2007-6890, presented at the AIAA Modeling and Simulation Technologies Conference, 20 August 2007, Hilton Head, South Carolina.

[ISO8601] : International Organization for Standards : "Data elements and interchange formats - Information interchange - Representation of dates and times [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40874&ICS1=1&ICS2=140&ICS3=30]" ISO 8601:2000, 2000

[Jackson04] : Jackson, E. Bruce; Hildreth, Bruce L.; York, Brent W.; and Cleveland, William : *Evaluation of a Candidate Flight Dynamics Model Simulation Standard Exchange Format [http://dscb.larc.nasa.gov/DCBStaff/ebj/Papers/aiaa-04-5038-DAVEdemo1.pdf]* . AIAA 2004-5038, presented at the AIAA Modeling and Simulation Technologies Conference, 17 August 2004, Providence, Rhode Island.

[Jackson02] : Jackson, E. Bruce; and Hildreth, Bruce L. : *Flight Dynamic Model Exchange using XML [http://techreports.larc.nasa.gov/ltrs/PDF/2002/aiaa/NASA-aiaa-2002-4482.pdf]* . AIAA 2002-4482, presented at the AIAA Modeling and Simulation Technology Conference, 5 August 2002, Monterey, California.

[Lin04] : Lin, Risheng; and Afjeh, Abdollah A. : “Development of XML Databinding Integration for Web-Enabled Aircraft Engine Simulation”. *J. Computing and Information Science and Engineering* 4 3 American Society of Mechanical Engineers September 2004

[NAA64] : North American Aviation : *Aerodynamic Data Manual for Project Apollo* SID 64-174C, 1964

[W3C-XLINK] : World Wide Web Consortium (W3C) : “W3C Recommendation: XML Linking Language (XLink) Version 1.0 [http://www.w3.org/TR/xlink/]” 2001-06-27

[W3C-XML] : World Wide Web Consortium (W3C) : “W3C Recommendation: Extensible Markup Language (XML) 1.0 [http://www.w3.org/TR/xml/]” 2008-11-26

[wiki01] : Combined wisdom of the Internet : “http://wikipedia.org/wiki/Spline_interpolation: "Spline Interpolation" [http://en.wikipedia.org/wiki/Spline_interpolation]”Cited 2006

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

X

Y

Z