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

Version 2.0RC1
$Revision: 273 $

AIAA Modeling and Simulation Technical Committee [http://www.aiaa.org/portal/index.cfm?GetComm=79&tc=tc]

$Id: DAVE-ML_ref.xml 273 2007-10-09 12:28:59Z bjax $

Abstract

This is a draft version of the eventual reference manual for DAVE-ML syntax and markup. DAVE-ML syntax is specified by the `DAVEfunc.dtd` Document Type Definition file; the version number above refers to the version of the `DAVEfunc.dtd.`

DAVE-ML is an open standard, being developed by an informal team of members of the American Institute of Aeronautics and Astronautics (AIAA). Contact the editor above for more information or comments regarding further refinement of DAVE-ML.

Table of Contents

Changes to this document

A list of changes during the course of development of the DAVE-ML Document Type Definition is given in this section.

Changes since version 1.9b3

- Added 'ceiling' and 'floor' enumeration selections to 'interpolate' attribute of `<independentVarPts>` and `<independentVarRef>` elements at the suggestion of Geoff Brian, Giovanni Cignoni, Randy Brumbaugh, and Daniel M. Newman.
- Added five uncertainty examples.
- Cleaned up all FIXME and BUG notes.
- Corrected and expanded the labels on the DAVE-ML excerpt figure.

Changes since version 1.9b2

- Corrected link to [Jackson04] paper.
- Added 'discrete' enumeration selection to 'interpolate' attribute of `<independentVarPts>` and `<independentVarRef>` elements at the suggestion of Geoff Brian, DSTO.
- Added section and `<variableDef>` example on extending the MathML 2 function set with `atan2`
- Removed all xns attributes from examples
- Amplified that it is a good practice to provide `<variableDef>`s in sorted sequence

Changes since version 1.8b1

- Added a `quadraticSpline` enumerated value to the `interpolate` attributes of the

`<independentVarPts>` and `<independentVarRef>` elements in response to a request from Geoff Brian of DSTO; fixed typo in `cubicSpline` attribute string. Added reference to Wikipedia article on spline interpolation [http://en.wikipedia.org/wiki/Spline_interpolation].

- Added a `classification` attribute to the `<reference>` element; added a `date` attribute to the `<modificationRecord>` element, per suggestions by Geoff Brian of DSTO.
- Added two-D and three-D ungridded table examples and figures; corrected typo on ungridded table definition syntax (thanks to Dr. Peter Grant of U. Toronto's UTIAS and Geoff Brian of Australia's DSTO).
- Reintroduced <!ENTITY> to include MathML2 DTD (complete) in body of this DTD. This entity definition quietly went away in version 1.6 due to a misunderstanding of proper way to include external DTDs; it is reintroduced to assist validating parsers.
- Added `<description>` sub-element to the `<provenance>` element, so the provenance entry can contain more information about change justification documents; made `<provenance>` or `<provenanceRef>` acceptable sub-elements to `<variableDef>` and `<checkData>` elements after a request from Geoff Brian of DSTO.

Changes since version 1.7b1

- Renamed `docID` attribute to `refID` of the `<modificationRecord>` so the attribute name is consistent; `docID` attribute is deprecated but remains for compatibility with older documents.
- Added `<correlatesWith>` and `<correlation>` sub-elements of `<uncertainty>` element to allow for multiple-dimensioned linear correlation of uncertainty of selected functions and variables.
- Added a new element, `<contactInfo>`, to replace the single `<address>` element. This format supports multiple ways to indicate the means of contacting the author of a document or reference. `<address>` is deprecated but is retained for backwards compatibility. This element also replaces the `email` and `xns` attributes of `<author>`.
- Fixed typographical error in `<ungriddedTableRef>` element: incorrect `gtID` attributed corrected to `utID`.
- Allowed multiple `<author>` elements wherever one was allowed before.
- Added a new tag, `<isStdAIAA/>`, to indicate a `variableDef` refers to one of the standard AIAA variables.
- Removed `<[un]griddedTable[Ref|Def]>` sub-elements of the `<confidenceBound>` element since this leads to circular logic.
- Changed SYSTEM ID to reflect new daveml.nasa.gov domain availability.
- Removed e-mail URLs to protect privacy of individual contributors.
- Added a new attribute, `interpolate`, to the `<independentVarPts>` element to indicate whether the table interpolation should be linear or cubic spline in the given di-

mension [modified to include quadratic in version 1.9].

- Added a new tag, `<isState/>`, to indicate a `variableDef` refers to a state variable in the model.
- Added a new tag, `<isStateDeriv/>`, to indicate a `<variableDef>` refers to a state derivative variable in the model.

Changes since version 1.6b1

Added `<checkData>` and associated elements. Added `<description>` sub-element to `<reference>` element.

Changes since version 1.5b3

Added `<uncertainty>` element. Emphasized MathML content markup over presentation markup. Several grammatical and typographical errors fixed; added figure 1. Added ISO 8601 (Dates and Times) reference.

Changes since version 1.5b2

- Added Bill Cleveland (NASA Ames' SimLab) and Brent York (NAVAIR's Manned Flight Simulator) to the acknowledgments section, to thank them for their pioneering initial trials of DAVE-ML.
- Added `<provenanceRef>` element and changed all parents of `<provenance>` elements to be able to use a `<provenanceRef>` reference instead (these were `<function>`, `<griddedTableDef>` and `<ungriddedTableDef>`) to eliminate duplicate `<provenance>` elements.

  Realization dawned that there was little difference between `<griddedTable>` and `<griddedTableDef>`s but the latter was more flexible (ditto `<ungriddedTable>` and `<ungriddedTableDef>`s). By making the `gtID` and `utID` attributes "implied" instead of "required," we can use the `Def` versions in both referenced-table and embedded-table `<function>`s. Thus the original `<griddedTable>` and `<ungriddedTable>` elements have been marked as "Deprecated." They are still supported in this DTD for backwards compatibility but should be avoided in future use; the easiest way to modify older DAVE-ML models would be to rename all `<griddedTable>`s as `<griddedTableDef>`s.

Changes since version 1.5b

- Fixed typos (thanks, Bill)!
- Added `<fileVersion>` element to `<fileHeader>` element, so each version of a particular DAVEfunc model can be uniquely identified. Format of the version identifier is undefined.
- Added an email attribute to the `<author>` element. The eXtensible Name Service (xns [http://www.xns.org/pages/xns_ov.html]) standard doesn't appear to be catching on as

rapidly as hoped, so a static e-mail link will have to do for now.

- Added a mandatory varID attribute to both `<independentVarPts>` and `<dependentVarPts>` so these can be associated with an input and output signal name (`<variableDef>`), respectively.
- Added an optional `<extraDocRef>` element to the `<modificationRecord>` element so more than one document can be associated with each modification event; if only one document needs to be referenced, use of the optional refID in the `<modificationRecord>` itself will suffice.

Introduction

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

Many benefits may be derived from application of XML in general, and DAVE-ML in particular, to the exchange of aerospace vehicle data:

- Human-readable, UNICODE text representation of the model
- Unambiguous machine-readable model description, suitable for conversion into programming language or direct import into object-oriented data structures
- The same source file can be used for computer-aided design and real-time piloted simulation
- Based on open, non-proprietary, standards that are language- and facility-independent
- Statistical properties, such as confidence bounds and uncertainty ranges, can be embedded, suitable for Monte Carlo or other statistical analysis of the model
- Compliant with AIAA draft simulation data standards
- Self-contained, complete, archivable data package, including references to reports, wind-tunnel tests, author contact information, data provenance
- 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.nasa.gov* [http://daveml.nasa.gov]

Purpose

DAVE-ML is intended to convey an entire flight vehicle dynamic simulation package, as is traditionally done with engineering development and flight training simulations. It is intended to allow 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 site-specific, and is often incomplete. Many times, the preparing facility makes assumptions about the knowledge the receiving facility has about the way the preparer's simulation environment is structured; these assumptions are not always true. As a result, the "re-hosting" of the dynamic flight model can take weeks if not months as the receiving facility staff gets their hands around the contents and arrangement of the data package, the model structure, the various data formats, and then spends additional time running check cases (if they are lucky enough to have received any) and tracking down small differences in implementations.

There are obvious benefits if this tedious, manual process could be somewhat automated. Often, when a paired set of facilities has exchanged one model, the receipt of another model is much faster, since the receiving facility will probably have built some computer scripts and processes to convert the data (both model and check-case data).

The purpose of DAVE-ML is to define a common exchange format for this data. The advantage gained is that any simulation facility or laboratory, after having written a DAVE-ML import and/or export script, could automatically receive and/or generate such packages (and updates to those packages) extremely quickly from other DAVE-ML-compliant facilities.

To accomplish this goal, the DAVE-ML project is starting with the bulkiest part of the most aircraft simulation packages: the aerodynamic model. This early version of DAVE-ML can be used to transport a complete aerodynamics model, including descriptions of the aerodynamic build-up equations and the data tables, as well as include references to the documentation about the aerodynamic 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 & inertia model, or a weapons load-out model, or perhaps a navigational database. The only requirement is that model outputs can be unambiguously defined in terms of inputs, with no past history information required.

Background

The idea of a universally-understood flight dynamics data package has been discussed for at least two decades, within the American Institute of Aeronautics and Astronautics (AIAA) technical committees. 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, came up with a combination 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. Some simulation standards have been proposed by the AIAA and are under active consideration at this writing.

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 & Simulation Technical Committee has prepared a draft standard for the exchange of simulation modeling data. This included a methodology for accomplishing the gradual standardization of simulation model components, a mechanism for standardizing variable names within math models, and proposed HDF as the data format. [AIAA01], [AIAA03]

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.

This proposal included several key aspects from the draft standard, including allowing use of the AIAA variable name convention, 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 results of a demonstration of using DAVE-ML to transmit two aerodynamic models between simulation facilities, showing the feasibility of the idea.

Supporting technologies

DAVE-ML relies on MathML, version 2.0, as a means to describe mathematical relationships. MathML is a low-level specification 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 home web page, found at http://www.w3.org/Math/.

MathML provides a fairly 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 arc tangent function which provides a continuous angle calculation by comparing the sine and cosine components of a two-dimensional coordinate set. Thus DAVE-ML provides a means to extend MathML 2 for a small predefined set of functions (currently only the 'atan2' function is supported). Thus, a DAVE-ML compliant processing tool should recognize this extension (which is accomplished using the valid MathML 2 'csymbol' element). See the variable definition element section for a discussion and an example of inserting an extension to MathML 2, the atan2 function, into a DAVE-ML `calculation` element.

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.

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

The `DAVEfunc` major element

The `DAVEfunc` element contains both data tables and equations for a particular vehicle subsystem model, for example, the aerodynamic model or the mass/inertia model. A `DAVEfunc` element is broken into roughly five components: a file header, variable definitions, breakpoint definitions, table definitions, and function definitions. This decomposition reflects common practice in engineering development flight simulation models in which the aerodynamic database is usually captured in multidimensional, linearly interpolated function tables. The input 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 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, calculations and outputs through the subsystem model. 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 of the model, and/or the results of intermediate calculations. Variables must be defined for each input and output for any function elements as well as any input or output of the subsystem represented. MathML [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. MathML *presentation* markup can also be used to define the symbol to use in documentation for each defined variable. 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. These sets are referenced by "gridded" function table definitions and may be referenced by more than one `function` definition.

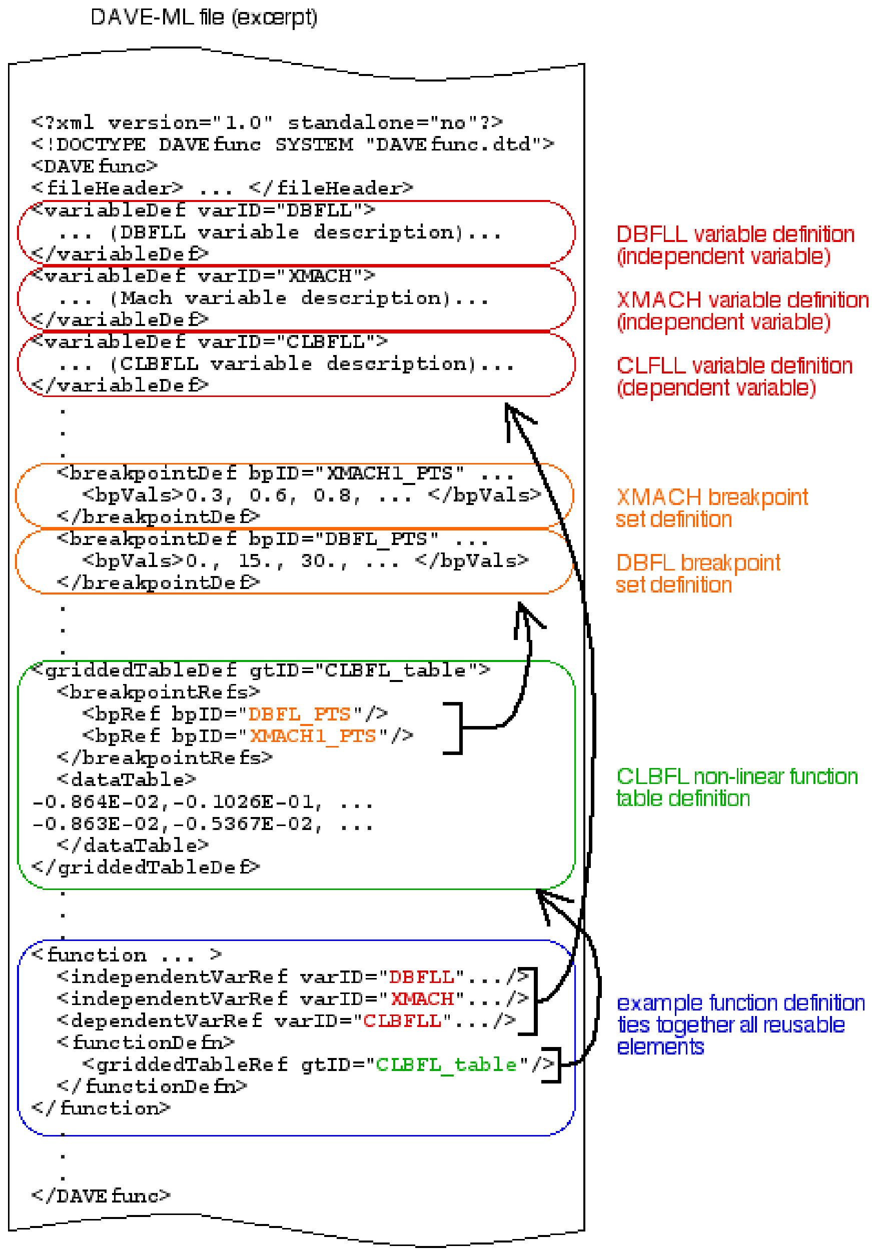
Function table definitions, described by `griddedTableDef` and `ungriddedTableDef` elements, generally contain the bulk of data points in an aero model, and typically represent a smooth hyper-surface representing the value of some aerodynamic non-dimensional coefficient as a function of one or more vehicle 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 `checkData` elements) 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, as well as optional internal signal values within the model to assist in debugging an instantiation of the model by the recipient.

Figure 1 contains excerpts from an example model, showing the 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.

Schematic overview of DAVEfunc

Shown below are schematic overviews of the various elements currently available in DAV-

Efunc. Each element is described in detail in appendix A. 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?
            [ choice_one_sub-element | choice_two_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 signal used within 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.

The variableDefs 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 functions are defined 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 at least one but can be used by more than one 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 more than one function definition if necessary for efficiency. Or all functions 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 values is important and must match the order given in the using function. Thus, functions that share an ungridded table 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. 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 described further below. Complete descriptions of each element is given in detail in appendix A.

The 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 a modification history.

```
fileHeader : [name]
    author : name, org, [email]
        contactInfo* :
    fileCreationDate : date
```

```
        fileVersion? :
            (version identification, character data)
        description? :
            (description of model, character data)
        reference* : refID, author, title, date, [accession,
href]
            description? :
                (description of reference,  character data)
        modificationRecord* : modID, [refID]
            author : name, org, [email]
                address? :
                    (address character data)
                description? :
                    (description of modification, character
data)
            extraDocRef? : refID
```

`fileHeader` sub-elements:

`author`	Name, organization, and optional email address of the author
`fileCreationDate`	Creation date of this file. See the "Additional DAVE-ML conventions" section later in this document for the recommended format.
`fileVersion`	A string that indicates the version of the document. No convention is specified for the format, but best practices would include an automated revision number from a configuration control process.
`description`	Optional but recommended text description: what does this DAVE-ML file represent?
`reference`	A list of zero or more references with a document-unique ID (must begin with alpha character), author, title, date, and optional accession and URL of the reference. Can include a description of the reference.
`modificationRecord`	An optional list of modifications with optional reference pointers, 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.

Example 1. An example of a `fileHeader` element

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

```
-->


  <fileHeader> ❷
    <author name="Bruce Jackson" org="NASA Langley Research
Center" email="nospam@nasa.gov">
      <address>MS 132 NASA, Hampton, VA 23681</address>
    </author>
    <fileCreationDate 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
     TM-107580. This aero model was used for HL-20 approach
and
     landing studies at NASA Langley Research Center during
1989-1995
     and for a follow-on study at NASA Johnson Space Center
in 1994
     and NASA Ames Research Center in 2001. This DAVE-ML
version
     created 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"
        email="nospam@nasa.gov">
        <address>MS 132 NASA, Hampton, VA 23681</address>
      </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 makes these big files more readable by humans.
❷ Start of `fileHeader` element.
❸ See the note regarding date format convention below.
❹ In this example, the revision number is automatically inserted by CVS or RCS, an automated versioning system.
❺ All documents referenced by notation throughout the file should be described here, in `reference` elements.
❻ All modifications made to the contents of this file should be given here for easy reference in separate `modificationRecord` elements.

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 XML-unique `varID` identifier (used for automatic code generation), 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. An 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. A final sub-element, `uncertainty`, captures the statistical properties of a (normally constant) parameter.

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]
        description? :
            (description character data)
        calculation? :
            math (defined in MathML2.0 DTD) :
        isOutput? :
        isStdAIAA? :
        uncertainty? : effect
            (normalPDF : numSigmas | uniformPDF : symmetry )
```

`variableDef` attributes:

`name`	A UNICODE name for the variable (may be same as the `varID`).
`varID`	An XML-legal name that is unique within the file.
`units`	The units-of-measure for the signal.
`axisSystem`	An optional indicator of the axis system (body, inertial, etc.) in which the signal is measured. See Conventions below for best recommended practice for nomenclature.
`sign`	An optional indicator of which direction is considered positive (+RWD, +UP, etc.). See the section on Conventions below,

for best recommended practice for abbreviations.

`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.

`variableDef` sub-elements:

`description` — An optional text description of the variable.

`calculation` — An optional container for the MathML 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].

`isOutput` — This optional element, if present, identifies 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 this `DAVEfunc` model.

`isStdAIAA` — This optional element, if present, signifies that this variable is one of the standard AIAA simulation variable names that are defined in the (draft) AIAA Simulation Standard Variable Names [AIAA01]. Such identification should make it easier for the importing application to hook up this variable (probably an input or output of the model) to the appropriate importing facility's signal.

`uncertainty` — This optional element, if present, describes the uncertainty of this parameter. See the section on Statistics below for more information about this element. Note that the `uncertainty` sub-element makes sense only for constant parameters (e.g., those with no `calculation` element but with an `initialValue` specified.

Example 2. Two examples of `variableDef` elements defining input signals

In the example below, two input variables are defined: `XMACH` and `DBFLL`. These two variables are inputs to a table lookup function shown in example 10 below.

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


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

```
  <variableDef name="MachNumber"❶ varID="XMACH"❷ units=""
symbol="M">
    <description> ❸
        Mach number (dimensionless)
    </description>
  </variableDef>

  <variableDef name="dbfll" varID="DBFLL" units="deg"❹
sign="+TED"❺
                symbol="&#x3B4;bfll"❻>
    <description>
        Lower left body flap deflection, deg, +TED (so
deflections are
        always zero or positive).
    </description>
    <isStdAIAA/>❼
  </variableDef>
```

❶ The `name` attribute is intended for humans to read, perhaps as the signal name in an automated wiring diagram. Note that "MachNumber" is a standard simulation parameter name.

❷ The `varID` attribute is intended for the processing application to read. This must be an XML-valid identifier and must be unique within this model.

❸ The `description` element may be used in an automated data dictionary entry associated with the `name` attribute.

❹ The optional `units` attribute describes the units of measure of the variable. See the section on Conventions below for a recommended list of units-of-measure abbreviations.

❺ 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 the section on Conventions below for a recommended list of sign abbreviations.

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

❼ 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 [AIAA01] or subsequent standards document

Example 3. A simple local variable

This example defines `CRBFLLO` which is the "independent variable" output from the table lookup function shown in example 10 below.

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

<!-- PRELIMINARY BUILDUP EQUATIONS -->

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

<!--    table output signal   -->
  <variableDef name="Cldbfll_0" varID="CRBFLL0" units="">
    <description>
```

```
        Output of CRBFLL0 function; rolling moment
contribution of
        lower left body flap deflection due to alpha^0
(constant
        term).
    </description>
  </variableDef>
```

Example 4. A more complete example using a `calculation` element

Here the local variable `CLBFLL` is defined as a calculated quantity, based on several other input or local variables (not shown). Note the `description` element is used to describe the equation, in FORTRANish 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="">
    <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>
        <apply>  ❸
        <plus/>
          <ci>CLBFLL0</ci>
          <apply>
            <times/>
            <ci>ALP</ci>
            <apply>
              <plus/>
              <ci>CLBFLL1</ci>
              <apply>
                <times/>
                <ci>ALP</ci>
                <apply>
                  <plus/>
                  <ci>CLBFLL2</ci>
                  <apply> ❹
                    <times/>
                    <ci>ALP</ci>
                    <ci>CLBFLL3</ci>
                  </apply> <!--            a*c3   -->  ❺
                </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 FORTRANish equation is simply for human readers and is not parsed by the processing application.
❷ A `calculation` element always embeds a MATHML-2 `math` element.
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 Inner-most apply multiplies variables `ALP` and `CLBFLL3`.
❺ The comments here are useful for humans to understand how the equation is being built up; the processing application doesn't use these comments.

Example 5. An output variable based on another `calculation` element

This 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="" sign="+UP"
symbol="CL">
    <description>
        Coefficient of lift
        CL = CL0 + CLBFUL + CLBFUR + CLBFLL + CLBFLR +
                   CLWFL + CLWFR + CLRUD + CLGE + CLLG
    </description>
    <calculation>
      <math>
        <apply> ❶
          <plus/>
          <ci>CL0</ci>
          <ci>CLBFUL</ci>
          <ci>CLBFUR</ci>
          <ci>CLBFLL</ci>
          <ci>CLBFLR</ci>
          <ci>CLWFL</ci>
          <ci>CLWFR</ci>
          <ci>CLRUD</ci>
          <ci>CLGE</ci>
          <ci>CLLG</ci>
        </apply>
      </math>
    </calculation>
    <isOutput/> ❷
  </variableDef>
```

❶ Here `<apply>` simply sums the value of these variables, referenced by their `varID`s.
❷ 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 extension (atan2) to the standard MathML

function set

In this example, we demonstrate a means to encode a non-standard MathML 2 math function, atan2. 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 `csymbol` element, specifically provided to allow extension of MathML 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; the first argument represents the sine
component and the second
      argument is the cosine component.
    </description>
    <calculation>
      <math>
        <apply>
          <csymbol
definitionURL="http://daveml.nasa.gov/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 example shows how to calculate wind roll angle, phi, from angle of attack and angle of sideslip; it comes from the Apollo aero data book [NAA64].

❷ The `csymbol` element is provided by MathML 2 as a means to extend the function set of MathML. Only a limited set of extensions given in this Standard are supported but others may be added to the standard in later versions. Note the specific URL that uniquely identifies this function; it is also the address of the documentation of the interpretation of the atan2 function.

❸ BETA would be the `varID` of a previously defined variable.

❹ ALPHA would be the `varID` of a previously defined variable.

The breakpoint set definition element

The breakpoint set definition element, `breakpointDef`, is used to define a list of comma-separate values that define the coordinate values along one axis of a gridded linear function value table. It contains a mandatory `bpID`, a file-unique XML identifier attribute, an optional `name` and units-of-measure attributes, an optional text `description` element and the comma-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-separated breakpoints)
```

`breakpointDef` attributes:

`bpID` An XML-legal name that is unique within the file.

`name` A UNICODE name for the set (may be same as `bpID`).

`units` The units-of-measure for the breakpoint values. See the section on Conventions below.

`breakpointDef` sub-elements:

`description` An optional text description of the breakpoint set.

`bpVals` A comma-separated, monotonically-increasing list of floating-point values.

Example 7. Two examples of `breakpointDef` elements

Two breakpoint sets are defined which are used in the `function` element given below (example 10). 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 7.

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


  <breakpointDef name="Mach" bpID="XMACH1_PTS" units="">  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">
    <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-separated list and must be in monotonically increasing order.

The gridded table definition element

The `griddedTableDef` element defines a multi-dimensional table of values corresponding with the value of an arbitrary function at the intersection of a set of specified independent inputs. 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-separated set of floating-point values. This list of values represents a multidimensional array whose size is inferred from the length of each breakpoint vector. For example, a two-dimensional 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 comma-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.

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.

```
    griddedTableDef* : [gtID, name, units]
        description? :
            (description character data)
        provenance? :
            author : name, org, [email]
                address? :
                    (address character data)
            functionCreationDate :
                (date in YYYY-MM-DD format, character data)
            documentRef* : docID
            modificationRef* : modID
        breakpointRefs :
          bpRef+ : bpID
        uncertainty? : effect
            (normalPDF : numSigmas | uniformPDF : symmetry )
        dataTable
          (character data)
```

`griddedTableDef` attributes:

`gtID` An XML-legal name that is unique within the file.

`name` A UNICODE name for the table (may be same as `gtID`).

`units` The units-of-measure for the table's output signal. See the section on Conventions 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`.

`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 the section on Statistics below 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-separated list. Parsing this list and storing it in the appropriate array representation is up to the implementor. By convention, the last breakpoint value increases most rapidly.

Example 8. An example of a `griddedTableDef` element

This non-linear function table is used by a subsequent `function` in example 9 to specify an output value based on two inputs 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 right lower 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"/>
      <functionCreationDate date="2003-01-31"/>
```

```
      <documentRef docID="REF01"/>
    </provenance>
    <breakpointRefs> ❺
      <bpRef bpID="DBFL_PTS"/>
      <bpRef bpID="XMACH1_PTS"/>
    </breakpointRefs>
    <dataTable> <!-- last breakpoint 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       -->
 0.29416E-01 , 0.34907E-01 , 0.38080E-01 , 0.41254E-01 ,
0.46014E-01 ,
 0.42215E-01 , 0.39681E-01 , 0.29547E-01 , 0.28211E-01 ,
0.19132E-01 ,
 0.16206E-01 , 0.14774E-01 , 0.12558E-01 ,
<!-- DBFL =       60.0       -->
 0.63779E-01 , 0.75685E-01 , 0.82566E-01 , 0.89446E-01 ,
0.99767E-01 ,
 0.85587E-01 , 0.76127E-01 , 0.38301E-01 , 0.36569E-01 ,
0.24800E-01 ,
 0.21007E-01 , 0.19151E-01 , 0.16278E-01
    </dataTable>
  </griddedTableDef>
```

❶ Comments are a good idea for human readers

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

❸ Descriptions are a good idea 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 example, 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 idea.

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 examples 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)
    provenance? :
        author : name, org, [email]
            address? :
                (address character data)
        functionCreationDate :
            (date in YYYY-MM-DD format, character data)
        documentRef* : docID
        modificationRef* : modID
    uncertainty? : effect
        (normalPDF : numSigmas | uniformPDF : symmetry )
    dataPoint+ :
```

`ungriddedTableDef` attributes:

`utID` A mandatory XML-legal name that is unique within the file

`name` An optional UNICODE name for the table (may be same as `gtID`.

`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`.

`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 it's input space. This element includes one coordinate for each function input variable. Parsing this information into a usable interpolative function is up to the implementor. By convention, the coordinates are listed in the same order that they appear in the using function.

Example 9. An example of an `ungriddedTableDef` element, encoding the data shown in figure 2.

This two-dimensional 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 doesn't 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.

Note that in this example, the Mach breakpoints are orthogonally spaced while the angle of attack breakpoints are not; this is still an 'ungridded' table since at least one independent variable has non-orthogonal spacing.

```
 <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
     <functionCreationDate 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 <!-- alfawdp, flap,
CLB--></dataPoint> ❸
        <dataPoint> 1.0  10.00  0.95 <!-- alfawdp, flap,
CLB--></dataPoint>
        <dataPoint> 1.0  12.00  1.12 <!-- alfawdp, flap,
CLB--></dataPoint>
        <dataPoint> 1.0  14.00  1.26 <!-- alfawdp, flap,
CLB--></dataPoint>
        <dataPoint> 1.0  15.00  1.32 <!-- alfawdp, flap,
CLB--></dataPoint>
        <dataPoint> 1.0  17.00  1.41 <!-- alfawdp, flap,
CLB--></dataPoint>
        <dataPoint> 5.0 -5.00  -0.55 <!-- alfawdp, flap,
CLB--></dataPoint>
        <dataPoint> 5.0  0.00  -0.03 <!-- alfawdp, flap,
CLB--></dataPoint>
        <dataPoint> 5.0  5.00   0.50 <!-- alfawdp, flap,
CLB--></dataPoint>
        <dataPoint> 5.0  10.00  1.02 <!-- alfawdp, flap,
CLB--></dataPoint>
        <dataPoint> 5.0  12.00  1.23 <!-- alfawdp, flap,
```

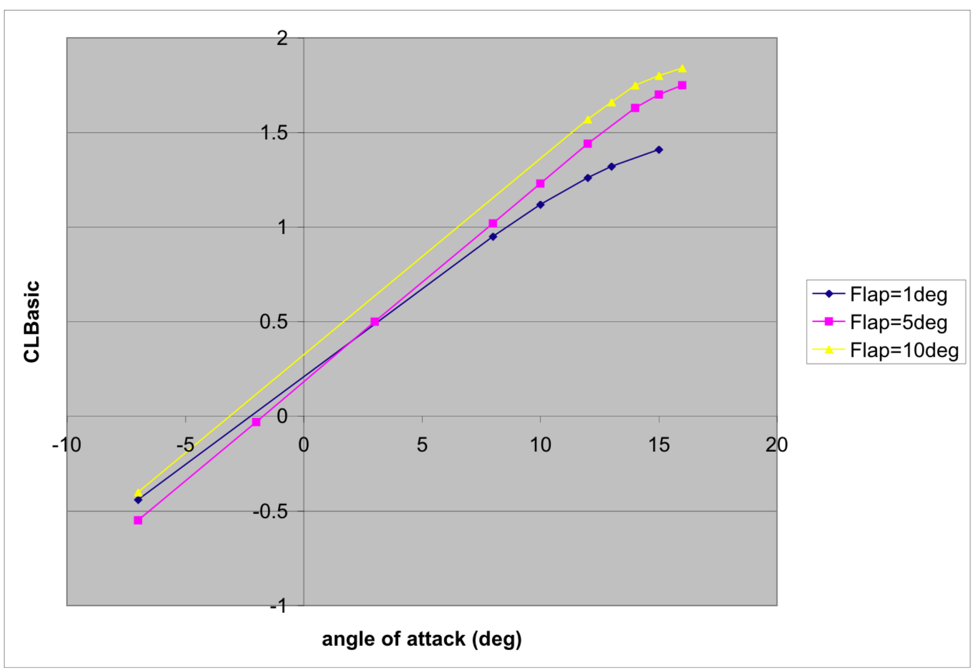
```
CLB--></dataPoint>
        <dataPoint> 5.0  14.00  1.44 <!-- alfawdp, flap, 
CLB--></dataPoint>
        <dataPoint> 5.0  16.00  1.63 <!-- alfawdp, flap, 
CLB--></dataPoint>
        <dataPoint> 5.0  17.00  1.70 <!-- alfawdp, flap, 
CLB--></dataPoint>
        <dataPoint> 5.0  18.00  1.75 <!-- alfawdp, flap, 
CLB--></dataPoint>
        <dataPoint> 10.0 -5.00 -0.40 <!-- alfawdp, flap, 
CLB--></dataPoint>
        <dataPoint> 10.0 14.00  1.57 <!-- alfawdp, flap, 
CLB--></dataPoint>
        <dataPoint> 10.0 15.00  1.66 <!-- alfawdp, flap, 
CLB--></dataPoint>
        <dataPoint> 10.0 16.00  1.75 <!-- alfawdp, flap, 
CLB--></dataPoint>
        <dataPoint> 10.0 17.00  1.80 <!-- alfawdp, flap, 
CLB--></dataPoint>
        <dataPoint> 10.0 18.00  1.84 <!-- alfawdp, flap, 
CLB--></dataPoint>

</ungriddedTableDef>
```

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

❷ Comments are ALWAYS a good idea for human readers

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

Figure 2. The two-dimensional lift function given in example 8



Example 10. An excerpt from a sample aero model giving an example of a three-dimensional `ungriddedTableDef` element, encoding the data shown in figure 3.

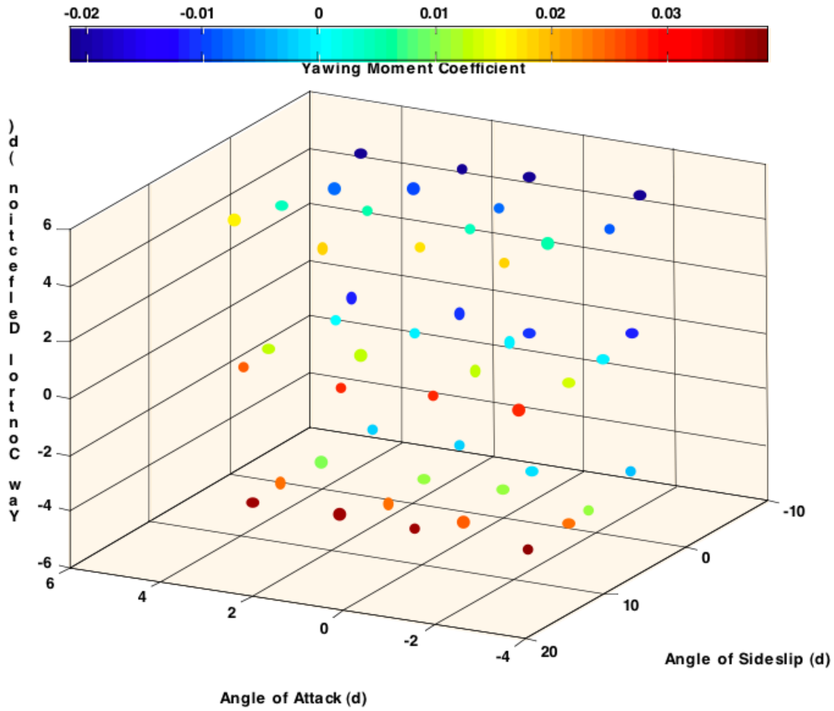
In this example, the dependent coordinates all vary in each dimension.

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

    <ungriddedTableDef name="yawMomentCoefficientTable1"
units="ND" utID="yawMomentCoefficientTable1">

       <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

Figure 3. The three-dimensional function given in example 9



The function definition element

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

```
    function* : name
        description? :
        provenance? :
            author : name, org, [email]
                address?
                    (address character data)
            functionCreationDate :
            extraDocRef* : docID
            modificationRef* : modID
        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
              {
                CHOICE OF
```

```
                {
                  griddedTableRef : gtID
                OR
                  griddedTableDef : [name]
                    breakpointRefs
                        bpRef+ : bpID
                    confidenceBound? : value
                    dataTable
                        (gridded data table as character
data)
                }
              }
              OR
              {
                CHOICE OF
                {
                  ungriddedTableRef : utID
                OR
                  ungriddedTableDef : [name]
                    confidenceBound? : value
                        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`.

`independentVarPts` If the author chooses, [he|she] can express a linearly-interpolated functions by specifying the independent (breakpoint) values sets as one or more `independentVarPts` which are comma-separated, monotonically increasing floating-point coordinate values corresponding to the `dependentVarPts` given next. 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` 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 a linear spline interpolation between points. The performance of interpolation of various orders is left up to the processing application. More information on relaxed spline interpolation may be found in [wiki01].

`dependentVarPts` — This element goes along with the previous element to specify a function table. Only one `dependentVarPts` may be specified. If the function is multidimensional, the convention is the last breakpoint dimension changes most rapidly in this comma-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 refer to separately-defined `variableDef`s. For multidimensional tables, the order of specification is important and must match the order in which breakpoints are specified or the order of coordinates in ungridded table coordinate/value sets.

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 a linear spline interpolation between points. The performance of interpolation of various orders is left up to the processing application. More information on relaxed spline interpolation may be found in [wiki01].

`dependentVarRef` — One `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.

`griddedTable` — As an alternative to reutilization of a previously defined table, this element may be used to define a private output gridded table. See the writeup on `griddedTableDef` for more information. [Deprecated: use of this element is discouraged and will not be supported in future DAVE-ML versions. Use a `griddedTableDef` instead.]

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

`ungriddedTable` — As an alternative to reuse of a previously defined table, this element may be used to define a private output ungridded table. See the writeup on `ungriddedTableDef` for more information. [Deprecated: use of this element is discouraged and will not be supported in future DAVE-ML versions. Use an `griddedTableDef` instead.]

Example 11. An 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 defined by the `CLBFL0_table` `griddedTableDef` (see example 7 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 function table. The processing application needs to pay attention to the `extrapolate` attribute, which details how to treat a variable whose value exceeds the stated limits on input.

❷ The dependent variable (XML name `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` that has an internal table

In this example, the function `CLRUD0` returns, in the variable `CLRUD0`, the value of function `CLRUD0_fn` represented by gridded table `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 15.0 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">
```

```
    <description>
        Rudder contribution to lift coefficient,
        polynomial multiplier for constant term.
    </description>
    <provenance> ❷
      <author name="Bruce Jackson" org="NASA Langley Research
Center" email="e.b.jackson@larc.nasa.gov"/>
      <functionCreationDate date="2003-01-31"/>
      <documentRef docID="REF01"/>
    </provenance>
    <independentVarRef varID="abs_rud"  min="0.0" max="15."
extrapolate="neither"/>
    <independentVarRef varID="XMACH" min="0.3" max="4.0"
extrapolate="neither"/>
    <dependentVarRef varID="CLRUD0"/>

    <functionDefn name="CLRUD0_fn">
      <griddedTableDef name="CLRUD0_table"> ❸
        <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 `provenance` element is required by the AIAA standard.
❸ This example has an embedded gridded table.

Example 13. A simple one-dimensional `function`

```
<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.

The checkData element

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

```
    checkData :
        staticShot* : name [refID]
            checkInputs :
                signal* :
                    signalName
                    [signalUnits]
                    signalValue
            [ internalValues :
                signal* :
                    signalID
                    signalValue ]
            checkOutputs :
                signal* :
                    signalName
                    [signalUnits]
                    signalValue
                    tol
```

`checkData` sub-elements:

`staticShot`	One or more check case data sets
`checkInputs`	A vector of input variables & values
`internalValues`	A vector of internal signal values [optional]
`checkOutputs`	A vector of output variables, including values & required matching tolerances

Example 14. An example of a check-case data set for a simple model

```
  <checkData>
    <staticShot name="Nominal" refID="NOTE1"> ❶
      <checkInputs>  ❷
        <signal>  ❸
```

```
          <signalName>True_Airspeed_f_p_s</signalName>
          <signalUnits>ft/sec</signalUnits>
          <signalValue> 300.000</signalValue>
        </signal>
        <signal>
          <signalName>Angle_of_Attack_deg</signalName>
          <signalUnits>deg</signalUnits>
          <signalValue>   5.000</signalValue>
        </signal>
        <signal>
          <signalName>s_Body_Pitch_Rate_rad_p_s</signalName>
          <signalUnits>rad/sec</signalUnits>
          <signalValue>   0.000</signalValue>
        </signal>
        <signal>
          <signalName>delta elevator</signalName>
          <signalUnits>deg</signalUnits>
          <signalValue>   0.000</signalValue>
        </signal>
        <signal>
          <signalName>Xcg</signalName>
          <signalUnits>fract</signalUnits>
          <signalValue>   0.250</signalValue>
        </signal>
      </checkInputs>
      <checkOutputs> ❹
        <signal>  ❺
          <signalName>CX</signalName>
          <signalValue>-0.00400000000000</signalValue>
          <tol>0.000001</tol>
        </signal>
        <signal>
          <signalName>CZ</signalName>
          <signalValue>-0.41600000000000</signalValue>
          <tol>0.000001</tol>
        </signal>
        <signal>
          <signalName>CLM</signalName>
          <signalValue>-0.04660000000000</signalValue>
          <tol>0.000001</tol>
        </signal>
      </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 variable values are given, constituting the input "vector."
❹ The checkOutputs element defines output variable values corresponding to the input vector.
❺ Note the included tolerance value, 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 internalValues specified

```
<checkData>
  <staticShot name="Skewed inputs">
    <checkInputs>
      <signal>
        <signalName>True_Airspeed_f_p_s</signalName>
        <signalUnits>ft/sec</signalUnits>\
        <signalValue> 300.000</signalValue>
      </signal>
      <signal>
        <signalName>Angle_of_Attack_deg</signalName>
        <signalUnits>deg</signalUnits>
        <signalValue>  16.200</signalValue>
      </signal>
      <signal>
        <signalName>s_Body_Pitch_Rate_rad_p_s</signalName>
        <signalUnits>rad/sec</signalUnits>
        <signalValue>  -0.760</signalValue>
        </signal>
      <signal>
        <signalName>delta elevator</signalName>
        <signalUnits>deg</signalUnits>
        <signalValue>   4.567</signalValue>
      </signal>
      <signal>
        <signalName>Xcg</signalName>
        <signalUnits>fract</signalUnits>
        <signalValue>   0.123</signalValue>
      </signal>
    </checkInputs>
    <internalValues> ❶
      <signal>
        <signalID>vt</signalID>
        <signalValue>300.0</signalValue>
      </signal>
      <signal>
        <signalID>alpha</signalID>
        <signalValue>16.2</signalValue>
      </signal>
      <signal>
        <signalID>q</signalID>
        <signalValue>-0.76</signalValue>
      </signal>
       .
       .       (similar internal values omitted)
       .

    </internalValues>
    <checkOutputs>
```

```
        <signal>
          <signalName>CX</signalName>
          <signalValue> 0.04794994533333</signalValue>
          <tol>0.000001</tol>
        </signal>
        <signal>
          <signalName>CZ</signalName>
          <signalValue>-0.72934852554344</signalValue>
          <tol>0.000001</tol>
        </signal>
        <signal>
          <signalName>CLM</signalName>
          <signalValue>-0.10638585796503</signalValue>
          <tol>0.000001</tol>
        </signal>
      </checkOutputs>
    </staticShot>
  </checkData>
```

❶ A dump of all model-defined variable values is included in this example to aide in debugging the implementation by the recipient.

Statistical information encoding

Statistical measures of variation of certain parameters and functions can be embedded in a DAVE-ML model. This information is captured in a `uncertainty` element, which can be referenced by `variableDef`, `griddedTableDef` and `ungriddedTableDef` elements.

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 do not have a dependency direction (that is, the statistical uncertainty of either parameter is specified in terms of the other one so the calculation order doesn't 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 :
```

```
          [correlatesWith : varID]
          [correlation : varID, corrCoef]
     OR
       uniformPDF : symmetric=['yes'|'no']
           bounds [, bounds]
```

`uncertainty` attributes:

`effect` Indicates, by choice of four enumerated values, how the uncertainty is modeled: as an additive, multiplicative, or percentage variation about the nominal value, or an 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 or #s) is specified. Note here multiplicative and absolute bounds don't 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 that 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
            symmetric uniform uncertainty band
        </description>
        <isOutput/>
        <uncertainty effect="absolute">    ❷
            <uniformPDF symmetric="yes">
```

```
                <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 desired, the value of CDo can vary uniformly between 0.001 and 0.010.

Example 17. 10% uncertainty applied to output variable with 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 non-linear one-dimensional function, but the uncertainty is applied downstream of the table.

```
<DAVEfunc>
    <fileHeader>
       .
       .
       .
    </fileHeader>
    <variableDef name="Alpha_deg" varID="Alpha_deg"
units="d"/>
    <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 symmetric="yes">
                <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 one-dimensional function of angle of attack #.

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 one-dimensional function, but the uncertainty is applied downstream of the table.

```
<DAVEfunc>
    <fileHeader>
      .
      .
      .
    </fileHeader>
    <variableDef name="Alpha_deg" varID="Alpha_deg"
units="d"/>
    <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 symmetric="yes">
                <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 one-dimensional `Cm_u` function table description.

❷ This function gives the nominal values of `Cm_u` as a one-dimensional function of angle of attack #.

Example 19. A one dimensional table, point-by-point, Gaussian

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

```
<DAVEfunc>
    <fileHeader>
        .
        .
        .
    </fileHeader>
    <variableDef name="Alpha_deg" varID="Alpha_deg"
units="d"/>
    <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-# (99.7%) confidence bounds.
❸ This table defines ±3-# bounds 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="Alpha_deg" varID="Alpha_deg"
units="d"/>
    <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 application of 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, 0.30 </dataTable>
            </griddedTableDef>
        </functionDefn>
    </function>
    <function name="Nominal Cm">
        <description> Nominal pitching moment values prior to
application of 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 which varies with angle of attack according to a non-linear one-dimensional table defined later. When performing parametric variations, `CL_u` can take on a multiplicative variation of up to 20% (3#) with a Gaussian distribution.

❷ This element indicates that the variation of pitching moment `Cm_u` correlates with the variation of lift coefficient.

❸ Pitching-moment coefficient has a nominal value which varies as a function of angle of attack, according to a non-linear one-dimensional table defined later. When performing parametric variations, `Cm_u` can take on a 30% variation (3#) with a Gaussian distribution.

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

Additional DAVE-ML conventions

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

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 & moments about this reference point and leave the transfer of moments to the center of mass to the equations of motion.

Decomposition of flight dynamic subsystems

It is recommended that a vehicle's flight dynamic 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.

Date format in DAVE-ML

The recommended way of representing dates in DAVE-ML documentation, especially date attribute and creation date elements, is numerically in the order YYYY-MM-DD. Thus, July 15, 2003 is given as 2003-07-15. This is to conform to ISO-8601 regarding date and time formats.

Common sign convention notation

The following list of sign convention notation is recommended for adoption. Note the sign convention for most quantities is already fixed by the AIAA Recommended Practice [AIAA92], so this is actually a list of abbreviations for typical sign conventions:

Common DAVE-ML sign convention notation

Acronym: +AFT
Meaning: Positive aft
Acronym: +ANR
Meaning: Positive aircraft nose right
Acronym: +ANU
Meaning: Positive aircraft nose up
Acronym: +CWFN
Meaning: Positive clockwise from north
Acronym: +DN
Meaning: Positive down
Acronym: +E
Meaning: Positive eastward
Acronym: +FWD
Meaning: Positive forward
Acronym: +LFT
Meaning: Positive left
Acronym: +N
Meaning: Positive northward
Acronym: +OUT
Meaning: Positive outward
Acronym: +POS
Meaning: Always positive
Acronym: +RCL
Meaning: Positive right of centerline
Acronym: +RT
Meaning: Positive right
Acronym: +RWD
Meaning: Positive right wing down
Acronym: +TED
Meaning: Positive trailing edge down
Acronym: +TEL
Meaning: Positive trailing edge left
Acronym: +THR
Meaning: Positive beyond threshold
Acronym: +UP
Meaning: Positive up

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 No Dimensions) or blank if the signal is a dimensionless quantity or flag.

Informally, this attribute can take on any reasonable abbreviation for a set of units that might be understandable by the intended audience, in any set of units (English or ISO). For greater re-usability, it is recommended that the set of measurements listed in the AIAA Standard for Flight Simulation Models, of which this document is a part, which also suggests how to encode superscripted powers of units (ft_s-1 for ft/sec, for example).

Planned major elements

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

- 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

Further information

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

The editors would like to acknowledge the contributions, encouragement and helpful suggestions from Jon Berndt (Jacobs Sverdrup), Brent York (Indra), Bill Cleveland (NASA Ames), Geoff Brian (Australian DSTO), J. Dana McMinn (NASA Langley), Peter Grant (UTIAS), Giovanni A. Cignoni (University of Pisa), Daniel M. Newman (Ball Aerospace), Hilary Keating (Fortburn Pty. Ltd.), Riley Rainey (SDS International), Jeremy Furtek (Delphi Research) and Randy Brumbaugh (Indigo Innovations).

References

[Jackson04] : Jackson, E. Bruce; Hildreth, Bruce L.; York, Brent W.; and Cleveland, Williams : *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.

[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

[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

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

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

A. Element references and descriptions

Name

`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.

Name

`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) - the eXtensible Name Service identifier for the author [Deprecated]

`email` (optional) - the e-mail address for the primary author [Deprecated]

Description

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

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' subelement as a single means to identify an individual or corporation in lieu of typical (and quickly dated) e-mail, phone, or address information. The author element itself is deprecated and should be replaced with the contactInfo element

Name

`bounds` — Describes limits or standard deviations of statistical uncertainties

Content model

```
bounds :
     (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[n] [un]griddedTableRef reference to an existing table definition, or a private [un]griddedTableDef, or a private table. In the case of formal table references or definitions, these tables define their own dependency, independent of the underlying random variable (whose nominal value is probably specified in a parent table definition). 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.

Possible parents

`normalPDF`
`uniformPDF`

Allowable children

`dataTable`
`variableDef`
`variableRef`

Name

`bpRef` — Reference to a breakpoint list

Content model

```
bpRef :  bpID
     EMPTY
```

Attributes

`bpID` - the internal XML identifier for a breakpoint set definition

Description

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

Possible parents

`breakpointRefs`

Allowable children

NONE

Name

`bpVals` — String of comma-separated values of breakpoints

Content model

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

Attributes

NONE

Description

bpVals is a set of breakpoints; that is, 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 two-dimensional coefficient value table.

Possible parents

`breakpointDef`

Allowable children

NONE

Name

`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, document-unique XMLname for the breakpoint set
`units`	(optional) - the units of measure for the breakpoint set

Description

A breakpointDef is where gridded table breakpoints are given. Since these are separate from function data, may be reused.

Possible parents

`DAVEfunc`

Allowable children

`description`
`bpVals`

Name

`breakpointRefs` — Reference to a breakpoint definition

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`

Name

`calculation` — Used to delimit a MathML v2 calculation

Content model

```
calculation :  xmlns:mathml2
     math [8]
```

Attributes

```
xmlns:mathml2
```

Description

Optional calculation element is MathML 2 content markup describing how the signal is calculated.

Possible parents

`variableDef`

Allowable children

`math` [8]

Name

`checkData` — Gives verification data for encoded model.

Content model

```
checkData :
     (
          (provenance? | provenanceRef?)
, staticShot*)
```

Attributes

NONE

Description

This top-level element is the placeholder for verification data of various forms. It will include static check cases, trim shots, and dynamic check case information. The checkData element may contain information about the history of the checkData information.

Possible parents

`DAVEfunc`

Allowable children

`provenance`
`provenanceRef`
`staticShot`

Name

`checkInputs` — Lists input values for check case

Content model

```
checkInputs :
     signal+
```

Attributes

NON
E

Description

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

Possible parents

`staticShot`

Allowable children

`signal`

Name

`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`

Name

`confidenceBound` — Defines the confidence in a function

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 placeholder 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.

Name

`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

Name

`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 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 or variable's immediate value will be used to calculate another function of variable's value.

Possible parents

`normalPDF`

Allowable children

NONE

Name

`correlation` — Indicates the linear correlation of this function 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` -

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) random value.

Possible parents

`normalPDF`

Allowable children

NONE

Name

`dataPoint` — Defines each point of an ungridded table

Content model

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

Attributes

`modID` (optional) - the internal XML 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 a ungridded table, the intepreting application is required to handle what may be unsorted data.

Possible parents

`ungriddedTableDef`
`ungriddedTable`

Allowable children

NONE

Name

`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. Values are comma or whitespace separated values. 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

Name

`DAVEfunc` — Root level element

Content model

```
DAVEfunc :
     (fileHeader, variableDef+, breakpointDef*,
griddedTableDef*, ungriddedTableDef*, function*, checkData*)
```

Attributes

NON
E

Description

Root element is DAVEfunc, composed of a file header element followed by 1 or more variable definitions and 0 or more break point definitions, gridded or ungridded table definitions, and function elements.

Possible parents

NONE - ROOT ELEMENT

Allowable children

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

Name

`dependentVarPts` — Defines output breakpoint values

Content model

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

Attributes

`varID` - the XML 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 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 don't share breakpoint or table values with other functions.

Possible parents

`function`

Allowable children

NONE

Name

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

Content model

```
dependentVarRef :  varID
     EMPTY
```

Attributes

`varID` - the internal XML 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

Name

`description` — Verbal description of an entity

Content model

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

Attributes

NONE

Description

optional description is free-form text describing something.

Possible parents

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

Allowable children

NONE

Name

`documentRef` — Reference to an external document

Content model

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

Attributes

`docID` (optional) - the internal XML identifier of a reference definition element

`refID` - the internal XML 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 it's use in the 'reference' element. This attribute will be removed in a future version of DAVE-ML.

Name

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

Content model

```
extraDocRef :  refID
     EMPTY
```

Attributes

`refID` - If an extraDocRef is used, the refID attribute is required.

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

Name

`fileCreationDate` — Gives date of creation of entity

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.

Possible parents

`fileHeader`

Allowable children

NONE

Name

`fileHeader` — States source and purpose of file

Content model

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

Attributes

`name` (optional) - the name of the file

Description

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

Possible parents

`DAVEfunc`

Allowable children

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

Name

`fileVersion` — Indicates the version of the document

Content model

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

Attributes

NON
E

Description

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

Possible parents

`fileHeader`

Allowable children

NONE

Name

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

Content model

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

Attributes

`name` - the name of this function

Description

Each function has optional description, optional provenance, and either a simple input/output 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`

Name

`functionCreationDate` — Date of creation of a function table

Content model

```
functionCreationDate :  date
     EMPTY
```

Attributes

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

Possible parents

`provenance`

Allowable children

NONE

Name

`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`

Name

`griddedTable` — Definition of a gridded table; associates breakpoint data with table data.

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.

Name

`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` (optional) - an internal, document-unique XMLname for the table

`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 is specified separately from the actual function declaration and requires an XML identifier attribute so that it may be used by multiple functions. The table data point values are specified as comma-separated values in 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.

Possible parents

`DAVEfunc`
`functionDefn`

Allowable children

`description`
`provenance`
`provenanceRef`
`breakpointRefs`
`uncertainty`
`dataTable`

Name

`griddedTableRef` — Reference to a gridded table definition

Content model

```
griddedTableRef :  gtID
     EMPTY
```

Attributes

`gtID` - the internal XML identifier of a gridded table definition

Possible parents

`functionDefn`

Allowable children

NONE

Name

`independentVarPts` — Simple definition of independent breakpoints

Content model

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

Attributes

`varID` - the XML id 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 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 midpoint 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.' This element is used for simple functions that don't share breakpoint or table values with other functions.

Possible parents

`function`

Allowable children

NONE

Name

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

Content model

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

Attributes

`varID` - the internal XML 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 (enumerated)

- `neither`
- `min`
- `max`
- `both`

`interpolate` (optional) - Interpolation flags for independent variable (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.' 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

Name

`internalValues` — An optional dump of internal model values for debugging checkcases.

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`

Name

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

Content model

```
isOutput :
     EMPTY
```

Attributes

NONE

Description

Optional isOutput element signals a variable that should be forced to be an output, even if it is used as an input elsewhere. Otherwise, using program should assume a signal defined with no calculation is an input; a signal defined with a calculation but not used elsewhere is an output; and a signal defined as a calculation and used subsequently in the model is an internal signal.

Possible parents

`variableDef`

Allowable children

NONE

Name

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

Content model

```
isState :
     EMPTY
```

Attributes

NONE

Description

Option isState element identifies this variable as a state variable in a dynamic model; this tells the implementation that this is the output of an integrator (for continous models) or a discretely updated state (for discrete models).

Possible parents

`variableDef`

Allowable children

NONE

Name

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

Content model

```
isStateDeriv :
     EMPTY
```

Attributes

NON
E

Description

Option isStateDeriv element identifies this variable as a state derivative variable in a dynamic model; this tells the implementation that this is the output of an integrator (for continous models only).

Possible parents

`variableDef`

Allowable children

NONE

Name

`isStdAIAA` — Flag to identify standard AIAA simulation variable

Content model

```
isStdAIAA :
     EMPTY
```

Attributes

NONE

Description

Optional isStdAIAA element identifies this variable is one of the [draft] standard AIAA variable names which should be recognizable exterior to this module, e.g. AngleOfAttack_deg. This flag should assist importing tools determine when an input or output should match a facility-provided signal name without requiring further information.

Possible parents

`variableDef`

Allowable children

NONE

Name

`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) - an optional document reference 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`

Name

`modificationRef` — Reference to associated modification information

Content model

```
modificationRef :  modID
     EMPTY
```

Attributes

`modID` - the internal XML identifier of a modification definition

Possible parents

`provenance`

Allowable children

NONE

Name

`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 subelement references other functions or variables that have a linear correlation to the current parameter or function. The correlation subelement 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`

Name

`provenance` — Describes origin or history of the associated information

Content model

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

Attributes

`provID` (optional) - This optional attribute allows provenance info to be cited elsewhere.

Description

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

Possible parents

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

Allowable children

```
author
functionCreationDate
documentRef
modificationRef
description
```

Name

`provenanceRef` — References a previously defined data provenance description.

Content model

```
provenanceRef :  provID
     EMPTY
```

Attributes

`provID` - the internal XML identifier for the previously defined provenance

Description

When the provenance of a set of several data is identical, the first provenance element may be given a provID and referenced by later data elements as a space-saving measure.

Possible parents

```
variableDef
griddedTableDef
ungriddedTableDef
function
checkData
```

Allowable children

NONE

Name

`reference` — Describes an external document

Content model

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

Attributes

`xmlns:xlink`	
`xlink:type`	
`refID`	- an internal, document-unique, XML 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) - an optional URL to an on-line copy of the document

Description

A reference element associates an external document with an ID making use of XLink semantics.

Possible parents

`fileHeader`

Allowable children

`description`

Name

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

Content model

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

Attributes

NON
E

Description

This element is used to document the name, ID, value, tolerance, and units of measure for checkcases. When used with checkInputs or checkOutputs, the signalName subelement must be present (since check cases are viewed from "outside" the model); when used in an internalValues element, the signalID subelement should be used to identify the signal by ID (which is the model-unique internal reference for each signal). When used in a checkOutputs vector, the tol element must be present.

Possible parents

```
checkInputs
internalValues
checkOutputs
```

Allowable children

```
signalName
signalUnits
signalID
signalValue
tol
```

Name

`signalID` — Gives the XML-valid, model-unique identifier of a varDef

Content model

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

Attributes

NONE

Description

Used inside a checkCase element to specify the input or output varID

Possible parents

`signal`

Allowable children

NONE

Name

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

Content model

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

Attributes

NON
E

Description

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

Possible parents

`signal`

Allowable children

NONE

Name

`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

Name

`signalValue` — Gives the value of a checkcase signal/variable

Content model

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

Attributes

NONE

Description

Used inside a checkCase element 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

Name

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

Content model

```
staticShot :  name,  [refID]
     (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

`checkInputs`
`internalValues`
`checkOutputs`

Name

`tol` — Specifies the tolerance of value matching for model verification

Content model

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

Attributes

NON
E

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.

Possible parents

`signal`

Allowable children

NONE

Name

`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

This optional 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`

Name

`ungriddedTable` — Definition of an ungridded set of function data

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.

Name

`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`	(optional) - an internal, document-unique XML name for the gridded table
`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 is specified separately from the actual function declaration and requires an XML identifier attribute so that it may be used by multiple functions.

Possible parents

`DAVEfunc`
`functionDefn`

Allowable children

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

Name

`ungriddedTableRef` — Reference to an ungridded table

Content model

```
ungriddedTableRef :  utID
     EMPTY
```

Attributes

`utID` - the internal XML identifier of a ungridded table definition

Possible parents

`functionDefn`

Allowable children

NONE

Name

`uniformPDF` — Defines a uniform (constant) probability density function

Content model

```
uniformPDF :  symmetric
     bounds+
```

Attributes

`symmetric` - Indicates whether the boundaries are symmetric (+/-x) or asymmetric (+x to -y). (enumerated)

- `yes`
- `no`

Description

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

Possible parents

`uncertainty`

Allowable children

`bounds`

Name

`variableDef` — Defines signals used in DAVE-ML model

Content model

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

Attributes

`name`	- the name of the signal being defined
`varID`	- an internal, document-unique XML name 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

Description

variableDef elements provide wiring information - that is, 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 subelement.

Possible parents

`DAVEfunc`
`bounds`

Allowable children

`description`
`provenance`
`provenanceRef`

```
calculation
isOutput
isState
isStateDeriv
isStdAIAA
uncertainty
```

Name

`variableRef` — Reference to a variable definition

Content model

```
variableRef :  varID
     EMPTY
```

Attributes

`varID` - the internal XML identifier of a previous variable definition

Possible parents

`bounds`

Allowable children

NONE