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

Version 1.5b3

Bruce Jackson, NASA Langley Research Center

`<e.b.jackson@nasa.gov>`

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 author above for more information or comments regarding further refinement of DAVE-ML.

Table of Contents

Changes to this document

Changes since version 1.5b2

- Added Bill Cleveland (NASA Ames' SimLab) and Brent York (NAVAIR's Manned Flight Simulator) to the acknowledgements 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 `griddedTableDefs` but the latter was more flexible (ditto `ungriddedTable` and `ungriddedTableDefs`). By making the `gtID` and `utID` attributes "implied" instead of "required," we can use the `Def` versions in both referenced-table and embedded-table `functions`. 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 `griddedTables` as `griddedTableDefs`.

Changes since version 1.5b

- Fixed typos (thanks, Bill)!

- Added `fileVersion` element to `fileHeader` element, so each version of a particular DAVE-func 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 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 "rehosting" 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 hardcopy documetation

A more complete discussion on the benefits and design of DAVE-ML can be found at the DAVE-ML web site: *http://dcb.larc.nasa.gov/utils/fltsim/DAVE* [http://dcb.larc.nasa.gov/utils/fltsim/DAVE/index.html]

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 "rehosting" 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 checkcase 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 would, 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 (lofty) goal, the DAVE-ML project is starting with the bulkiest part of the most aircraft simulation packages: the aerodynamic model. This initial version of DAVE-ML can 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. 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 loadout model, or perhaps a navigational database. The only requirement is that model outputs can be unambigously 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 standarize 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 persuing such a goal.

This proposal includes several key aspects from the draft standard, including allowing use of the AIAA variable name convention, data table schema, and including tracability for each data point back to a referenced document or change order.

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/

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*, which are used to route inputs, calculations and outputs through the subsystem model are 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] markup is used to define constant, intermediate, or output variables in mathematical combination of constant values, function table outputs, and other variables. MathML 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 "dependent variable" in a function definition) so the output of functions can also be used as inputs to other variables or functions.

Breakpoint definitions, captured in `breakpointDef` elements, consist of a list of montomically-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 (usually) smooth hypersurface 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 used in several functions, such as a left- and right-aileron moment contribution.

Finally, 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.

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 reusability 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 `DAVEfunc`. Each element is described in detail in the appendix. 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*
```

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.
breakpointDef	A DAVEfunc model can contain zero, one or more breakpoint set definitions. These definitions can be shared among several gridded function tables.
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 necessary for efficiency. Or some or all functions in a model can specify their tables internally with an embedded `griddedTableDef` element or

without sharing any table definitions.

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) are not stored within the gridded table definition but are referenced by the parent function. This allows a function to be shared by more than one set of breakpoint values (such as left and right aileron deflections).

The associated function element that references a gridded table associates the dimensions of the table with both the correct breakpoint (independent coordinate) set and the corresponding independent variable or input signal for each dimension.

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.

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. Functions also include provenance, or background history, of the function data such as wind tunnel test or other source information.

Each of these sub-elements is described further in this reference, and 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, [xns, email]
        address? :
            (address character data)
    fileCreationDate : date
```

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

fileHeader sub-elements:

author	Name, organization, and optional XNS ID and mailing 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 reached, but this could include an automated revision number from configuration control processes.
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.
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"
        xns="@bjax" email="e.b.jackson@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"
    xns="@bjax" email="e.b.jackson@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 description and a mathematical description, in MathML 2 markup, of the calculations needed to derive the variable from other variables or function table outputs. A final optional sub-element, isOutput, serves to indicate an intermediate calculation that should be brought out to the

rest of the simulation.

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]
    description? :
        (description character data)
    calculation? :
        math (defined in mathML2.0 DTD) :
    isOutput? :
```

variableDef attributes:

name	A UNICODE name for the table (may be same as `gtID`.
varID	An XML-legal name that is unique within the file
units	The units-of-measure for the signal.
axsiSystem	An optional brief indicator of the axis system (body, inertial, etc.) in which the signal is measured
sign	An optional brief indicator of which direction is considered positive (+RWD, +UP, etc.)
symbol	A UNICODE Greek symbol for the signal [to be superceded with more formal MathML or Tex element in a later release]
varID	An XML-legal name that is unique within the file

variableDef sub-elements:

description	An optional text description of the variable
calculation	An optional container for the the MathML 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.

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

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

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

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

  <variableDef name="Mach"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;bf
    <description>
        Lower left body flap deflection, deg, +TED (so deflections are
        always zero or positive).
    </description>
  </variableDef>
```

❶   The `name` attribute is intended for humans to read, perhaps as the signal name in an automated
     wiring diagram.
❷   The `varID` attribute is intended for the processing application to read. This must be an XML-val-
     id 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 when 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.

In this example, two input variables are defined: XMACH and DBFLL. These two variables are inputs to
a table lookup function shown in example [FIXME] below.

Example 3. A simple local variable

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

This example defines CRBFLLO which is the "independent variable" output from the table lookup function shown in example [FIXME] below.

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

```
<!--     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>
                    <times/>
                    <ci>ALP</ci>
                    <ci>CLBFLL3</ci>
                  </apply> <!--              a*c3    -->  ❹
                </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`) opertor) and the arguments to that operation (in this case, a variable CLBFLL defined elsewhere is added to the results of the nested `apply` operation).
❹ The comments here are useful for humans to understand how the equation is being built up; the processing application doesn't use these comments.

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 FOR-

TRANish human-readable text. The calculation element describes this same equation in MathML 2 syntax; this portion should be used by parsing applications to create either source code, documentation, or run-time calculation structures.

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

```
    <!-- ================= -->
    <!--  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>
          <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/> ❶
  </variableDef>
```

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

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

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 `gtID`.

units   The units-of-measure for the breakpoint values.

`breakpointDef` sub-elements:

description   An optional description of the breakpoint set.

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

Example 6. Two examples of `breakpointDef` elements

```
<!--                          ====================                           -->
<!-- =========================  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 ascending order.

Two breakpoint sets are defined, which are used in the `function` element example given below (example 8). Breakpoint sets XMACH1_PTS and DBFL_PTS contain values for Mach and lower body flap deflection, respectively, for which function values are given in several function tables; one example is given below in example 7.

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 coordin-

ates along each dimension are defined in separate breakpointDef elements that are referenced within this element by bpRefs, one for each dimension.

The data contained within the dataTable 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 confidenceBound scalar value may be provided that reprents the confidence level in the values presented. [This statistical capability needs significant improvement in later DAVEfunc releases.]

```
griddedTableDef* : [gtID, name, units]
    description? :
        (description character data)
    provenance? :
        author : name, org, [xns, email]
            address? :
                (address character data)
        functionCreationDate :
            (date in YYYY-MM-DD format, character data)
        documentRef* : docID
        modificationRef* : modID
    breakpointRefs :
      bpRef+ : bpID
    confidenceBound? : value
    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.

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 bpRefs is important.

confidenceBound   The numeric value specified in this element represents the statistical con-
                       fidence in the values contained within this table.

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 rep-
                       resentation is up to the implementor. By convention, the last breakpoint
                       value increases most rapidly.

Example 7. An example of a `griddedTableDef` element

```
        <!-- ================================= -->  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" xns="@bjax"/>
      <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 auto 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 `bpRefs` are in the same order as the table is wrapped (see note 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.

This non-linear function table is used by a subsequent `function` in example 8 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 at.

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.

An optional confidenceBound scalar value may be provided that reprents the confidence level in the values presented. [This statistical capability needs significant improvement in later DAVEfunc releases.]

```
ungriddedTableDef* : [utID, name, units]
    description? :
        (description character data)
    provenance? :
        author : name, org, [xns, email]
            address? :
                (address character data)
        functionCreationDate :
            (date in YYYY-MM-DD format, character data)
        documentRef* : docID
        modificationRef* : modID
    confidenceBound? : value
    dataTable+ :
```

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`.

confidenceBound   The numeric value specified in this element represents the statistical confidence in the values contained within this table.

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.

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, [xns, email]
            address?
                (address character data)
        functionCreationDate :
        extraDocRef* : docID
        modificationRef* : modID
    EITHER
    {
      independentVarPts+ : varID, [name, units, sign, extrapolate]
          (input values as character data)
      dependentVarPts : varID, [name, units, sign]
          (output values as character data)
    }
    OR
    {
      independentVarRef+ : varID, [min, max, extrapolate]
      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:

nam e	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 specifing the indendent (breakpoint) values sets as one or more `inde-pendentVarPts` which are comma-separated, monotonically increasing coordinate values corresponding to the `dependentVarpts` given next. In the case of multiple dimensions, more than one `indep-dendentVarPts` must be specified, one for each dimension. The mandatory varID attribute is used to connect each independentVarPts with an input signal.
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 output values. The mandatory varID attribute is used to connect this table output to an output signal.
independentVarRef	One or more of these elements refer to a separately-defined `vari-ableDef`. For multidimensional tables, the order of specification is important and match the order in which breakpoints are specified or the order of coordinates in ungridded table coordinate/value sets.
dependentVarRef	One `dependentVarRef` must be specified to marry the output of the function to a particular `variableDef`.
functionDefn	This mandatory element (if not a simple function) identifies either a separately-specified data table 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.

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.

Example 8. An example of a function which refers to a previously defined `griddedTableDef`

```
    <!-- ============================= -->
    <!-- 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 flag, 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.

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).

Example 9. A `function` that has an internal table

```
    <!-- ================ -->
    <!-- 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" xns="@bjax"/>
          <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.

In this example, the function CLRUD0 returns, in the variable CLRUD0, the value of function CL-RUD0_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.

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 we've seen.

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 suggested since such dates sort with minimal effort, and is a common format on UNIX computer systems.

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

Lots more to identify

[FIXME: more conventions are lurking]

...like how to define units-of-measure notation

Planned major elements

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

- Validation check case definitions & data files

- Dynamic elements

Further information

Further information, background, the latest `DAVEfunc.dtd` and example models of some aircraft data packages can be found at the DAVE-ML web site: http://dcb.larc.nasa.gov/utils/fltsim/DAVE/index.html

References

[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://dcb.larc.nasa.gov/utils/fltsim/DAVE/SimParNames_Dec2001.pdf] ", Preliminary Draft, December 2001

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

A. Element references and descriptions

A description of each element of DAVE-ML is given below.

Element list

`address` - Street address or other contact information of an author
`author` - Gives name and contact information for originating party of the associated data
`bpRef` - Reference to a breakpoint list
`bpVals` - String of comma-separated values of breakpoints
`breakpointDef` - Defines breakpoint sets to be used in model

`breakpointRefs` - Reference to a breakpoint definition
`calculation` - Used to delimit a MathML v2 calculation
`confidenceBound` - Defines the confidence in a function
`dataPoint` - Defines each point of an ungridded table
`dataTable` - Gives a name to a table of function data
`DAVEfunc` - Root level element
`dependentVarPts` - Defines output breakpoint values
`dependentVarRef` - Identifies the signal to be associated with the output of a function
`description` - Verbal description of an entity
`documentRef` - Reference to an external document
`extraDocRef` - Allows multiple documents to be associated with a single modification event
`fileCreationDate` - Gives date of creation of entity
`fileHeader` - States source and purpose of file
`fileVersion` - Indicates the version of the document
`function` - Defines a function by combining independent variables, breakpoints, and tables.
`functionCreationDate` - Date of creation of a function table
`functionDefn` - Defines a function by associating a table with other information
`griddedTable` - Definition of a gridded table; associates breakpoint data with table data.
`griddedTableDef` - Defines an orthogonally-gridded table of data points
`griddedTableRef` - Reference to a gridded table definition
`independentVarPts` - Simple definition of independent breakpoints
`independentVarRef` - References a predefined signal as an input to a function
`isOutput` - Flag to identify non-obvious output signals from model
`modificationRecord` - To associate a reference single letter with a modification event
`modificationRef` - Reference to associated modification information
`provenance` - Describes origin or history of the associated information
`provenanceRef` - References a previously defined data provenance description.
`reference` - Describes an external document
`ungriddedTable` - Definition of an ungridded set of function data
`ungriddedTableDef` - Defines a table of data, each with independent coordinates
`ungriddedTableRef` - Reference to an ungridded table
`variableDef` - Defines signals used in DAVE-ML model

Name

address -- Street address or other contact information of an author

```
address
```

Content model

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

Attributes

NONE

Possible parents

```
author
```

Allowable children

NONE

Name

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

`author`

Content model

```
author : name, org, [xns], [email]
     address?
```

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

Description

author includes alternate means of identifying author using XNS or normal e-mail/address

Possible parents

```
fileHeader
modificationRecord
provenance
```

Allowable children

`address`

Name

bpRef -- Reference to a breakpoint list

`bpRef`

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

```
bpVals
```

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

```
breakpointDef
```

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

```
breakpointRefs
```

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

```
calculation
```

Content model

```
calculation :
      math
```

Attributes

NONE

Description

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

Possible parents

```
variableDef
```

Allowable children

```
math
```

Name

confidenceBound -- Defines the confidence in a function

`confidenceBound`

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.

Possible parents

```
griddedTableDef
ungriddedTableDef
griddedTable
ungriddedTable
```

Allowable children

NONE

Name

dataPoint -- Defines each point of an ungridded table

```
dataPoint
```

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 -- Gives a name to a table of function data

```
dataTable
```

Content model

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

Attributes

NONE

Description

The dataTable element is used by gridded tables where the indep. variable values are implied by break-point 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.

Possible parents

```
griddedTableDef
griddedTable
```

Allowable children

NONE

Name

DAVEfunc -- Root level element

```
DAVEfunc
```

Content model

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

Attributes

NONE

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
```

Name

dependentVarPts -- Defines output breakpoint values

```
dependentVarPts
```

Content model

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

Attributes

`varID` - the XML ide 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

```
dependentVarRef
```

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

```
description
```

Content model

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

Attributes

NONE

Description

optional description is free-form text describing something.

Possible parents

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

Allowable children

NONE

Name

documentRef -- Reference to an external document

```
documentRef
```

Content model

```
documentRef :   docID
      EMPTY
```

Attributes

`docID` - the internal XML identifier for of a reference definition element

Possible parents

`provenance`

Allowable children

NONE

Name

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

```
extraDocRef
```

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

```
fileCreationDate
```

Content model

```
fileCreationDate :  date
      EMPTY
```

Attributes

`date` - The date of the file, in YYYY-MM-DD format

Description

fileCreationDate is simply a string with a date in it.

Possible parents

```
fileHeader
```

Allowable children

NONE

Name

fileHeader -- States source and purpose of file

```
fileHeader
```

Content model

```
fileHeader :  [name]
      (author, fileCreationDate, fileVersion?, description?, reference*, modificationReco
```

Attributes

name (optional) - the name of the file

Description

The header element requires an 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

```
fileVersion
```

Content model

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

Attributes

NONE

Description

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

Possible parents

```
fileHeader
```

Allowable children

NONE

Name

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

```
function
```

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

```
functionCreationDate
```

Content model

```
functionCreationDate :  date
     EMPTY
```

Attributes

`date` - the creation date of the function, in YYYY-MM-DD format

Possible parents

```
provenance
```

Allowable children

NONE

Name

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

```
functionDefn
```

Content model

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

Name

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

```
griddedTable
```

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

```
griddedTableDef
```

Content model

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

Possible parents

```
DAVEfunc
```

Allowable children

```
description
provenance
provenanceRef
breakpointRefs
confidenceBound
dataTable
```

Name

griddedTableRef -- Reference to a gridded table definition

```
griddedTableRef
```

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

```
independentVarPts
```

Content model

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

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. 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

```
independentVarRef
```

Content model

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

Description

An independentVarRef more fully describes the input mapping of the function by pointing to a separate breakpoint definition element. This allows common breakpoint values for many tables.

Possible parents

```
function
```

Allowable children

NONE

Name

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

```
isOutput
```

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

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

`modificationRecord`

Content model

```
modificationRecord :  modID,  [refID]
     (author, description?, extraDocRef*)
```

Attributes

`modID` - a single letter used to identify all modified data with this mod
`refID` (optional) - an optional document reference for this modification

Description

A modificationRecord associates a single letter (such as modification "A") with a modification author, 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

`modificationRef`

Content model

```
modificationRef :  modID
     EMPTY
```

Attributes

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

Possible parents

`provenance`

Allowable children

NONE

Name

provenance -- Describes origin or history of the associated information

```
provenance
```

Content model

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

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
griddedTableDef
ungriddedTableDef
function
```

Allowable children

```
author
functionCreationDate
documentRef
modificationRef
```

Name

provenanceRef -- References a previously defined data provenance description.

```
provenanceRef
```

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

```
griddedTableDef
ungriddedTableDef
function
```

Allowable children

NONE

Name

reference -- Describes an external document

`reference`

Content model

```
reference :  xmlns:xlink,  xlink:type,  refID,  author,  title,  [accession],  date,
       EMPTY
```

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
`accession` (optional) - the accession number (ISBN or organization report number) of the document
`date` - the date of the document, in 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

NONE

Name

ungriddedTable -- Definition of an ungridded set of function data

```
ungriddedTable
```

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

```
ungriddedTableDef
```

Content model

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

Allowable children

```
description
provenance
provenanceRef
confidenceBound
dataPoint
```

Name

ungriddedTableRef -- Reference to an ungridded table

```
ungriddedTableRef
```

Content model

```
ungriddedTableRef :  gtID
     EMPTY
```

Attributes

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

Possible parents

```
functionDefn
```

Allowable children

NONE

Name

variableDef -- Defines signals used in DAVE-ML model

```
variableDef
```

Content model

```
variableDef :  name,  varID,  units,  [axisSystem],  [sign], [alias], [symbol], [
     (description?, calculation?, isOutput?)
```

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. Note the breakpoint values are specified separately since one input signal may be normalized to more than one breakpoint set (for gridded function data).

Possible parents

```
DAVEfunc
```

Allowable children

```
description
calculation
isOutput
```