Using the Proposed AIAA Aerodynamic Modeling Standard (AeroML)

Or

How to build a portable sim

August 2006

Bruce Jackson
NASA Langley Research Center
Hampton, Virginia

Outline

- Background
- History of sim model standards
- Underpinnings
- AeroML elements
- Examples
- Existing tools

Background

- Independent development of sim labs
- Incomplete standards
- Increasing reliance on simulation for R&D/Procurement
- Increased collaboration and teaming
- Possibilities for increased productivity
- *Portable* simulation models is the goal

History of model standards efforts

- MODSIM
- SIMNET/WARNET - DIS - HLA
- Project 2851 - SEDRIS
- Internal NASA: NASP project

MODSIM

- Air Force initiative in mid-80s to standardize hardware/software configurations for aircraft simulations
- Early proposal was based on Gould 3277's
- Evolved into Modular Aircrew Simulation System (MASS) using VME boards to distribute simulation models
- Forerunner of present-day JMASS?

SIMNET - DIS - HLA

- Late 1980's DARPA push to connect multiple DoD simulations via Ethernet (Jack Thorpe)
- SIMNET developed by BBN to exchange state data; used 'dead reckoning' to predict ahead with frequent updates on state
- Became IEEE 1278.1 Distributed Interactive Simulation (DIS) standard in 1995
- DMSO replaced DIS with High-Level Architecture (HLA) standard in late 90's

JMASS

- Joint Modeling and Simulation System
- Developed in early 90s to provide graphical way to develop simulation models that ran within specified computing environment
- Uses UML to describe object-oriented model components
- Specifies standard I/O for models
- Promotes reuse (of JMASS and legacy) models

SEDRIS

- Synthetic Environment Data Representation and Interchange Specification (SEDRIS)
- Developed to promote reuse of environmental data (terrain, obstructions, weather) in networked simulations (visual databases)
- Grew out of DoD project 2851:
  Standard Simulator Data Base (SSDB)
  Interchange Format (SIF) and
  Standard Simulator Data Base Facility (SDBF)
- See MIL-STD-1821

NASA NASP Sim Standards

- Developed by Larry Schilling of NASA Dryden c. 1988
- Specified FORTRAN as standard language
- Specified standard variable names, table formats, time history data formats
- Specified axes, sign conventions, and units of measure

Flight Dynamic Model Standards

- Few widely recognized flight sim standards exist!
- 1992 AIAA/ANSI standard established common axis systems and Greek notation [AIAA/ANSI R-004-1992]
- Bruce Hildreth called for standard variable names and data table formats in 1998 AIAA Flight Sim Conference (Boston) [AIAA 98-4576]
- In 2002 paper, Jackson and Hildreth outlined possible approach using XML; estimated potential savings at $ 6.9 million for one tactical aircraft (across 59 different simulations) [AIAA 2002-4482]
- Jackson, Hildreth, et. al. reported successful aero model exchange between NASA and NAVAIR in 2004 [AIAA 2004-5038]

Why flight dynamic model standards?

- Growth of teaming within aero industry
- Exponential increase in desktop compute power
- Increasing reliance on sim-based acquistions
- Emergence of web-based everything

- Simulation community is not, however, an early adopter…

DAVE-ML vs. AeroML

- Long-term effort, Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML)
  - Encode model's static and dynamic elements
  - Constitute complete math model of flight dynamics
  - Portable simulation model with verification data
- Near-term effort, Aerodynamic Markup Language (AeroML), encodes bulk of flight dynamics
  - Data tables
  - Buildup equations
  - Uncertainty bounds
  - Static verification shots

Selected implementation - why XML?

- Open standard (W3C)
- Transportable
- Human- and machine-readable UNICODE
- Programming language neutral
- Good fit for databases and models
- Transformable (->HTML, etc.) via XSLT
- Lots of generic XML tools, editors, parsers
- Takes advantage of existing MathML (math markup)

Underpinnings - quick XML intro

- Extensible Markup Language (adopted 1998 by the World-Wide-Web Consortium,W3C)
- Derived from SGML (Standardized General ML)
- Related to HTML
  - but HTML is NOT valid SGML
  - XHTML is an XML/HTML hybrid
- Various ‘applications’ or grammars using XML have been developed
- AeroML uses MathML 2.0 for buildup equations
- Ubiquitous - Microsoft® Office documents

Simple XML file example


```
<?xml version="1.0"?>
<EudoraStats>
    <version>2</version>
    <startTime>Wed, 10 Apr 2002 14:36:23 -0600</startTime>
    <currentTime>Sun, 20 Aug 2006 07:36:48 -0600</currentTime>
    <receivedAttachments>
        <current>
          <day>1</day>
          <week>1</week>
          <month>59</month>
          <year>1391</year>
        </current>
        <averageSum>
          <day>994</day>
          <week>400</week>
          <month>75</month>
          <year>12</year>
        </averageSum>
        <total>6276</total>
    </receivedAttachments>
</EudoraStats>
```

Schemas / DTDs

- XML content layout or schemas have two main flavors:
  - Document Type Definitions (DTDs) (older style)
  - XML Schema Documents (XSDs) (newer style)
- Other schemas also exist: RelaxNG, etc.
- Schemas are *not* required for informal data encoding
- Schemas *are* required for widely shared data
  (it represents a contract between provider and user)
- Some concern in XML community that XSDs are overkill
- AeroML specified by more widely understood DTD

Key AeroML content

- Non-linear function descriptions
  - Tabular (traditional aero models)
    - Unlimited dimensionality
    - Orthogonal grid or unstructured points
  - Polynomial or exponential expressions
- Model build-up equations (using MathML)
- Units of measure
- Standard variable names
- Uncertainty bounds
- Provenance (history/source of data)
- Checkcases (verification data)

Current AeroML limitations (opportunities!)

- Static models (dynamics not yet supported)
- Hierarchy of models not yet supported
- Need a native AeroML editor
  - Typically generated with Perl or other scripts
  - Implies hand-editing with generic text or XML editor
- Draft AIAA standards document soon
  - Variable names
  - Units of measure and combined dimensional units

Similar project - JSBSim

- Open-source effort to provide standard EOM and simulation framework (J. Berndt)
- Entire simulation specified in XML grammar
  - Some limitations relative to AeroML, but a good start
  - Includes mass, inertia, control system elements
- Preliminary GUI modeling tool available
- Probably can convert parts of models between JSBSim XML format and AeroML
- See http://jsbsim.org

Benefits of AeroML

- Makes available full-fidelity aerodynamic models from dissimilar simulation S/W frameworks
- Supports automatic verification of received models
- Compatible with legacy code frameworks
- Supports Monte Carlo analysis
- Includes heritage or provenance of model
- Avoids proprietary modeling languages
- Provides human- and computer-readable format
- Allows gradual progress toward run-time loading

Existing AeroML examples

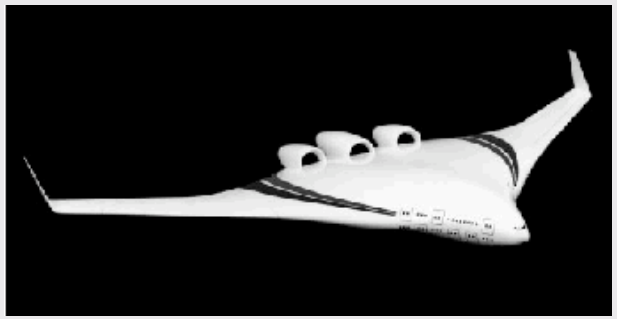
Fighter subsonic aero model

- 51 variables, 18 tables, 744 points
- Switches & absolute value nonlinear elements
- 17 verification checkcases included
- 154 KB file with 2,712 lines

Concept development lifting body aero model

- Supersonic and subsonic regimes
- 361 variables, 168 tables, 6,240 points
- 24 verification checkcases included
- 1.2 MB file with 22,299 lines

Other AeroML uses



NASA/Boeing Blended-wing-body (X-48A)

- Complete aero model in 12.5 MB text file
- 22 breakpoint sets, 97 tables (up to 5D)
- 256 functions using 716,826 data points
- Compresses to 2.6 MB
- Parsed in 5 seconds on average PC

Boeing X-37 air-launched test vehicle

- AeroML used as intermediate format
- Boeing data in Excel tables
- Langley simulation in Simulink
- Generated multiple Simulink models, one per Excel file

Other AeroML uses

Orion launch abort vehicle
(CEV with and w/o escape tower)

- 46,332 data points, 3 inputs, 8 outputs
- 385KB text file
- AeroML used as an intermediate format between Johnson Space Center and Langley Research Center analysis tools

Existing (known) AeroML resources

- DAVE-ML Reference manual (website)
- JANUS (C++ library)
  - Australian DSTO/Ball Aerospace
- NASA Ames FTP tool (import/export Perl scripts)
- NAVAIR's CASTLE sim shell (latest release)
- XSLT conversion script
  AeroML => XHTML
- DAVEtools (Java packages):
  AeroML => Simulink

XSLT conversion script

- eXtensible Stylesheet Language Transformation
- XSLT scripts, themselves XML documents, specify how to convert XML-based files into other formats.
- 'DAVE_html.xsl' details a conversion from AeroML XML file into an XHTML document
- Requires *jdom*, Apache *xerces* or similar tool
- Available for download =>
  http://daveml.nasa.gov

DAVEtools 0.5.3

- Two Java (1.4 or higher) packages
  - Base package, "DAVE"
    - Builds internal objects from AeroML description
    - Performs auto-verification of model
    - Allows simple exercising of model
  - Simulink® export tool, "DAVE2SL" (built on DAVE)
    - Converts AeroML models into Simulink blocks
- Java source code available
  - Request public domain license via http://daveml.nasa.gov

Janus API Library

- Developed by Australia's Defence Science & Technology Org (DSTO) (G. Brian)
- Janus is a C++ library to read/write and manipulate DAVE-ML files
- Reads DAVE-ML/AeroML directly at run-time
- AES-256 (!) encryption for classified models
- Associated Matlab code to read/write DAVE-ML
- Moving toward public domain release

Store-ML & Carna API library

- Also developed by Australian DSTO (G. Brian)
- Extends DAVE-ML for use with ‘stores’ or weapons load-out configuration
- Handles mass/inertia models

Demonstrations

- Generate HTML
- Generate Simulink® subsystem
- Exercise command line DAVEtool

AeroML grammar

- Top-level syntax
- Specified by DAVEfunc.dtd Document Type Def
- Examples taken from reference manual
- Reference manual available for download from http://daveml.nasa.gov - see "DTDs"

Top-level syntax

- Top level-element is <DAVEfunc>

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

Key: *element : attribute [optional attribute]*
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

File header element

- Provides documentation trail for model

```
fileHeader : [name]
        author : name, org
        fileCreationDate : date
        fileVersion? :
        description? :
        reference* :
        modificationRecord* :
        provenance* :
```

Key: *element : attribute [optional attribute]*
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Example of `fileHeader` element

Opening of `fileHeader` element

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

Author information

```
<fileHeader>
    <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"/>
```

Model description subelement

```
    <fileVersion>$Revision: 1.24 $</fileVersion>
    <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 AERO-ML version
       created 2003 by Bruce Jackson to demonstrate AERO-ML.
    </description>

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

We define `refID`' s here;
used elsewhere

`fileHeader` *example, cont'd*


```
  <reference refID="REF02"
    author="Cleveland, William F
    title="Possible Typo in HL
    accession="email"
    date="2003-08-19"
  />

  <modificationRecord modID="A" refID="REF02">
    <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>
```

Modification record defines `modID` for easy reference later, where modification has been applied

Note citation of `refID`

Ending fileHeader tag

Variable Definition element (`variableDef`)

- Used to define each variable used in the model
  - Includes inputs, outputs, constants, parameters and local (temporary) variables

```
variableDef : name, varID, units,
        [axisSystem, sign, alias, symbol, initialValue]
    description? : (description character data)
    calculation? : math (defined in MathML2.0 DTD)
    isOutput? :
    isStdAIAA? :
    uncertainty? : effect (additive | multiplicative | percentage | absolute)
        (normalPDF : numSigmas | uniformPDF : symmetric )
```

Key: element : attribute [optional attribute]
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Example of `variableDef` *for two input elements*

Standard Name flag

Name for documentation

Symbol for documentation

Internal name

Sign convention

Units

Note UNICODE symbol

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

<variableDef name="MachNumber" varID="XMACH" units="" symbol="M">
  <description>
        Mach number (dimensionless)
  </description>
  <isStdAIAA/>
</variableDef>

<variableDef name="dbfll" varID="DBFLL" units="deg" sign="ted" symbol="∂bfll">
  <description>
        Lower left body flap deflection, deg, +TED (so deflections are
        always zero or positive).
  </description>
</variableDef>
```

Example of local variable definition

Simple definition; used later

```
<!-- 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>
```

A more complex variable definition


```
<!--    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))
    </description>
    <calculation>
      <math>
        <apply>
          <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   -->
                </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>
```

This MathML2 encodes:

$$C_{L\partial BFLL} = C_{L\partial BFLL0} + \alpha C_{L\partial BFLL1} + \alpha^2 C_{L\partial BFLL2} + \alpha^3 C_{L\partial BFLL3}$$

An output variable definition


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

Normal `variableDef` preamble

Sum of inputs

Output flag

Breakpoint set definition element (`breakpointDef`)

- Used to define a common set of independent variable values associated with one or more tables of function data

```
breakpointDef : bpID, [name, units]
        description? : (description character data)
        bpVals : (list of comma-separated set of breakpoint values)
```

Key: element : attribute [optional attribute]
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Two examples of `breakpointDef` *elements*

Breakpoint values

Breakpoint ID

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

<breakpointDef na    Mach" bpID="XMACH1_PTS" units="">
  <description>
    Mach number brea  oints 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,
  </bpVals>
</breakpointDef>
```

Breakpoint ID

```
<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>
```

Breakpoint values

Gridded function table definition

- Defines nonlinear function dependent values

```
griddedTableDef : [gtID, name, units]
    description? : (description character data)
    provenance? :
        author : name, org, [xns, email]
            address? : (address character data)
        functionCreationDate
        documentRef* : docID
        modificationRef* : modID
    breakpointRefs :
        bpRef+ : bpID
    uncertainty? : effect (additive | multiplicative | percentage | absolute)
        (normalPDF : numSigmas | uniformPDF : symmetric )
    dataTable : (list of comma-separated set of table values)
```

Key: element : attribute [optional attribute]
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Example of `griddedTableDef`

Header information

```
<griddedTableDef name="CLBFL0" gtID="CLBFL0_table">
  <description>
    Lower body flap contribution to lift coefficient, polynomial constant term
  </description>
  <provenance>
    <author name="Bruce Jackson" org="NASA Langley Research Center"/>
    <functionCreationDate date="2003-01-31"/>
    <documentRef docID="REF01"/>
  </provenance>
  <breakpointRefs>
    <bpRef bpID="DBFL_PTS"/>
    <bpRef bpID="XMACH1_PTS"/>
  </breakpointRefs>
  <dataTable> <!-- last breakpoint changes most rapidly -->
    <!--  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    -->
-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 , etc.
    </dataTable>
</griddedTableDef>
```

Breakpoint set references

Data set values

An aside about table ordering…

- The appearance of tables in AERO-ML files can be confusing in that they appear to be 2-dimensional
- In actuality, the values are a 1-D vector of the 'unraveled' n-dimensional tables with last index changing fastest
- Example: given $C_L = C_L(\alpha, M, \delta f)$, value sequence is

  `<datatable>`

  $C_L(\alpha_1, M_1, \delta f_1)$, $C_L(\alpha_1, M_1, \delta f_2)$, … $C_L(\alpha_1, M_1, \delta f_n)$,
  $C_L(\alpha_1, M_2, \delta f_1)$, $C_L(\alpha_1, M_2, \delta f_2)$, … $C_L(\alpha_1, M_2, \delta f_n)$,
  …
  $C_L(\alpha_1, M_m, \delta f_1)$, $C_L(\alpha_1, M_m, \delta f_2)$, … $C_L(\alpha_1, M_m, \delta f_n)$,
  $C_L(\alpha_2, M_1, \delta f_1)$, …

  `</datatable>`

- Line breaks can occur anywhere - they are unimportant

Aside - table ordering (cont'd)

- Previous example could also be encoded

<datatable>

$C_L(\alpha_1, M_1, \delta f_1)$, $C_L(\alpha_1, M_1, \delta f_2)$,
$C_L(\alpha_1, M_1, \delta f_n)$, $C_L(\alpha_1, M_2, \delta f_1)$,
$C_L(\alpha_1, M_2, \delta f_2)$, … $C_L(\alpha_1, M_2, \delta f_n)$,
…
$C_L(\alpha_1, M_m, \delta f_1)$,
$C_L(\alpha_1, M_m, \delta f_2)$, … $C_L(\alpha_1, M_m, \delta f_n)$,
$C_L(\alpha_2, M_1, \delta f_1)$, …

</datatable>

Aside - table ordering (concluded)

- Order of `<bpRefs>` in `<breakpointRefs>` element is important!
- In previous example, order would have to be

```
<breakpointRefs>
      <bpRef bpID="ALPHA_PTS"/>
      <bpRef bpID="MACH_PTS"/>
      <bpRef bpID="DELTA_FLAP_PTS"/>
</breakpointRefs>
```

…so interpreter will know which breakpoint set to associate with which dimension of this function table.

- This is one of the few places in AeroML where order is important.

Function definition element

- This element ties breakpoint sets to tables
- Several different syntaxes; below is most common

```
function : name
    description? : (description character data)
    provenance? :
    independentVarRef+ : varID, [min, max, extrapolate]
    dependentVarRef : varID
    functionDefn : [name]
        griddedTableRef : gtID
```

Key: element : attribute [optional attribute]
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Example of `function` *element*


```
    <!-- ============================== -->
    <!-- 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"/>
  <independentVarRef varID="XMACH" min="0.3" max="4.0" extrapolate="neither"/>
  <dependentVarRef varID="CLBFLL0"/>
  <functionDefn name="CLBFL0_fn">
    <griddedTableRef gtID="CLBFL0_table"/>
  </functionDefn>
</function>
```

We define a function `CLBFLO_fn` to be

`CLBFLO_fn = CLBFLO_fn(DBFLL, XMACH)`

with limits on input values and no extrapolation

Verification data (`checkData`) element

Contains input/output vector pairs (time slices)

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

Key: element : attribute [optional attribute]
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Example checkData dump

Checkcase identifier

Checkcase input vector

name

units

value

```
<checkData>
  <staticShot name="Nominal" refID="NOTE1">
    <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>   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>
        .
        .     (other input values)
        .
    </checkInputs>
      .
      .
      .
```

Example `checkData` *element (cont'd)*

Checkcase output vector

```
      .
      .
      .
    <checkOutputs>
      <signal>
        <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>
        .
        .     (other output values & tolerances)
        .
    </checkOutputs>
  </staticShot>
      .
      .     (other input/output vector pairs)
      .
</checkData>
```

`checkData` *elements can have intermediate values to assist debugging*


```
<checkData>
  <staticShot name="Skewed inputs">
    <checkInputs>
      .
      .    (checkcase input values)
      .
    </checkInputs>
    <internalValues>
      <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>
          .
          .    (additional internal values; normally one for every internal variable)
          .
    </internalValues>
    <checkOutputs>
      .
      .    (checkcase output values)
      .
    </checkOutputs>
  </staticShot>
</checkData>
```

Checkcase intermediate values

`uncertainty` *Element*

(every project should, and probably does, have one)

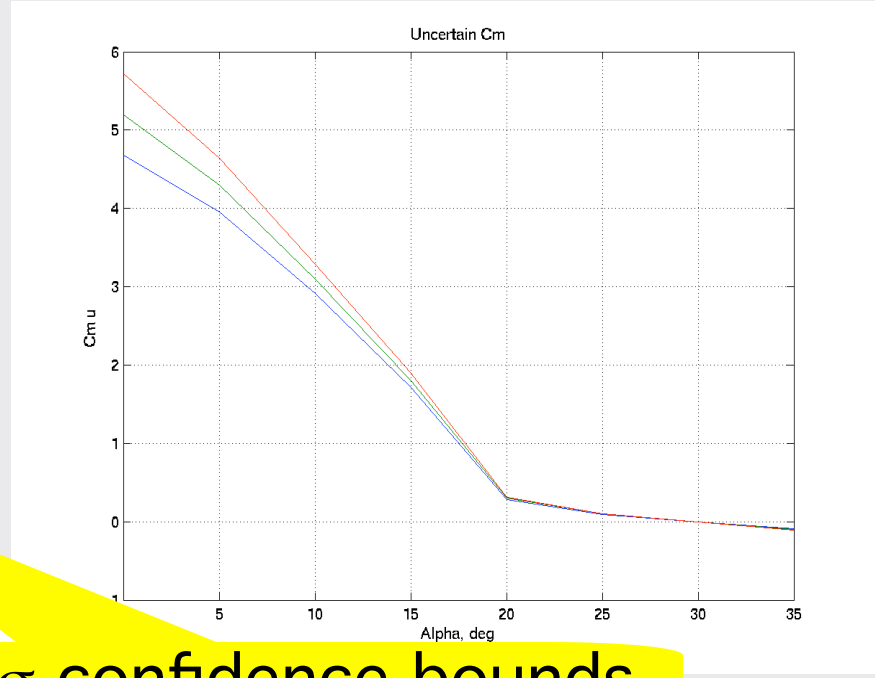
```
uncertainty : effect=['additive'|'multiplicative'|'percentage'|'absolute']
   EITHER
    normalPDF : numSigmas=['1','2','3',…]
         bounds : (value corresponding to 1, 2, 3,… σ)
   OR
    uniformPDF : symmetric=['yes'|'no']
         bounds : (symmetric or lower bound value)
         [bounds : (upper bound value)]
```

Key: *element : attribute [optional attribute]*
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Application of `uncertainty` *element*

Example: add 5-12% uncertainty bound around C_m

```
<function name="Uncertain Cm">
  <independentVarRef varID="Alpha_deg"/>
  <dependentVarRef varID="Cm_u"/>
  <functionDefn>
    <griddedTableDef>
      <breakpointRefs>
        <bpRef bpID="ALP"/>
      </breakpointRefs>
      <uncertainty effect="percentage">
        <normalPDF numSigmas="3">
          <bounds>
            <dataTable>
  0.10, 0.08, 0.06, 0.05, 0.05,
  0.06, 0.07, 0.12
            </dataTable></bounds>
        </normalPDF>
      </uncertainty>
      <dataTable>
  5.2, 4.3, 3.1, 1.8, 0.3, 0.1, 0.0,
      </dataTable>
    </griddedTableDe
  </functionDefn>
</function>
```



3 σ confidence bounds

(assuming ALP = [0:5:35])

Nominal function values

Summary

- AERO-ML sufficient for some production models
  - static models only, such as aero, mass properties
- Limited tools available, but slowly growing
  - Chicken or egg problem right now
  - Hand-coding not recommended - write export script
- Syntax will evolve, in backwards-compatible way
- Still to tackle: dynamics & subsystems
- Help wanted! Join the mailing list - see website

http://daveml.nasa.gov