Evaluation of a Candidate Flight Dynamics Model Simulation Standard Exchange Format

E. Bruce Jackson*
NASA Langley Research Center, Hampton, VA 23681

Bruce L. Hildreth†
SAIC, Lexington Park, MD 20653

Brent W. York‡
Naval Air Systems Command, Patuxent River, MD 20670

and

William B. Cleveland§
Northrop Grumman Information Technology, Moffett Field, CA 94035

This paper reports on a study that tested the AIAA Modeling and Simulation Technical Committee's proposed simulation model exchange standard. The motivation for and results of an evaluation of using the standard in an XML-based exchange format for static aerodynamic models between simulation facilities is given. The format is described briefly, along with a description of the two fixed-wing aerodynamic models used in testing. The experiences by two facilities that volunteered to import these test models are given, along with a description of the development of preliminary export tools. Utility programs developed to assist in implementing the standard format are described.

Nomenclature

Cx	=	force coefficient in the *x* body axis direction
Cy	=	force coefficient in the *y* body axis direction
Cz	=	force coefficient in the *z* body axis direction
Cl	=	moment coefficient about the *x* body axis
Cm	=	moment coefficient about the *y* body axis
Cn	=	moment coefficient about the *z* body axis

Partial Acronyms List

CASTLE	Controls Analysis and Simulation Test Loop Environment
CPAN	Comprehensive Perl Archive Network
DAVE-ML	Dynamic Aerospace Vehicle Exchange Markup Language
DTD	Document Type Definition (an XML schema encoding method)
FTP	Function Table Processor
Unicode	The universal character glyph encoding scheme
VMS	NASA Vertical Motion Simulator
XHTML	an XML-based version of the world-wide web's Hypertext Markup Language
XML	eXtensible Markup Language (human- and machine-readable data encoding scheme)
XPath	XML Path description syntax

* Senior Aerospace Technologist, Dynamics & Control Branch, MS 132, AIAA Associate Fellow.
† Vice President/Division Manager, Suite 200, 22299 Exploration Drive, AIAA Senior Member.
‡ Aerospace Engineer, AIAA Senior Member.
§ System Engineer.

I. Introduction

IN the early to mid 1990s the AIAA Modeling and Simulation Technical Committee embarked on development of a standard format for the exchange of flight dynamics models. The objective of this standard is to facilitate the exchange of an air vehicle model between simulation facilities. The concept behind this standard is to be a comprehensive and consistent format for import and export of models between facilities. It includes a standard syntax to describe function tables, mathematical equations, functional block diagrams, and verification test data that each simulation facility would use when sharing a model with another facility. The receiving facility would thus have to be able to interpret one standard format, instead of the unknown data format(s) of the exporting facility. This allows each facility to create and maintain only one set of export and import tools. Also, each facility would not have to change any of their internal software architecture; they just have to be able to import models from and export models to the standard format.

In the late 1990s a candidate format was developed for the aerodynamics portion of the model, primarily the function tables, mathematical equations and the definitions needed for clear exchange of the information[1].

Progress in actual implementation of the standard format has been slow until the last year. Two of the authors of this paper (Jackson and Hildreth) proposed an implementation[2] of the aerodynamics portion of the standard using eXtensible Markup Language (XML)[*,3]. An aerodynamic model typically contains the largest amount of data in a flight dynamic model and was considered the easiest component to encode using a specialized application of XML. Function table data were encoded with application-specific tags; the buildup equations were encoded using MathML version 2, a set of mathematical XML tags[†,4]. This encoding scheme was named the Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML)[‡]. The specification of DAVE-ML is given as a Document Type Definition (DTD). This specification evolved as a direct result of the activity reported here; as of this writing, version 1.7b1 of the DAVE-ML DTD is current and available for evaluation. Some comments below refer to earlier versions.

XML has several advantages over proprietary data formats, including being both machine- and human-readable text. New features are easily added for a particular application, and XML editing templates are available for a number of different programming environments[§,**,††,‡‡].

To evaluate this proposed realization of the standard, a small group of AIAA members from NASA Ames Research Center, Naval Air Systems Command (NAVAIR) at Patuxent River, and NASA Langley Research Center embarked on a demonstration of DAVE-ML. They developed initial import tools for transferring aerodynamic models from the standard format and have been testing the format by importing two fixed-wing aerodynamic models from NASA Langley to NAVAIR Patuxent and NASA Ames using DAVE-ML. Additionally, they have extended the XML-based standard to include static check data so that the model can be automatically verified after import.

This paper reports upon this activity, discusses the lessons learned in exchanging the models through the standard format, and discusses the benefits of using DAVE-ML. The recent progress is considered a major step in the implementation of a useful standard format benefiting the industry in general.

II. Example simulations

To familiarize the demonstration participants with DAVE-ML, a generic fighter subsonic aerodynamics model originally from NASA[5], popularized by Stevens & Lewis[6] and later expanded and encoded into Matlab scripts by Garza & Morelli[7] was described using DAVE-ML and provided to the participants as a pathfinder. An excerpted version is found in the Appendix to this report.

This aerodynamics model contains 51 variables, four breakpoint sets, 18 one- and two-dimensional tables, and a total of 744 data points. It defines outputs for six aerodynamic coefficients (*Cx*, *Cy*, *Cz*, *Cl*, *Cm*, *Cn*) as function of angles of attack and sideslip, Mach number, angular body rates & airspeed. It includes non-linear function

[*] http://www.xml.org/ [cited July 5, 2004]
[†] http://www.w3c.org/TR/MathML [cited July 5, 2004]
[‡] http://dcb.larc.nasa.gov/utils/fltsim/DAVE [cited July 5, 2004]
[§] http://www.oxygenxml.com/ [cited July 5, 2004]
[**] http://www.xmlcooktop.com/ [cited July 5, 2004]
[††] http://www.exchangerxml.com/ [cited July 5, 2004]
[‡‡] http://www.xmlhack.com/read.php?item=2061 [cited July 5, 2004]

definitions as interpolated tables, switches, and absolute value elements in the variable definitions. It includes 17 input-output checkcases for automatic verification.

This model was verified against an independent Simulink implementation of the same fighter model that was developed in an unpublished effort by John Davidson of NASA Langley Research Center.

A second, more complex aerodynamic model of a conceptual lifting-body vehicle was also provided for testing; this 1.2 MB, 22,300 line HL-20 aerodynamics model contained 6,240 data points in 168 tables and 24 input-output checkcases for verification of the model by the implementing facility. It was a more representative aerodynamics model of the type used in high-fidelity engineering simulation studies.

These models were provided to team members at NASA Ames and NAVAIR Patuxent for testing of their prototype import tools. Their experiences in decoding and using these models are documented below.

III. Implementation Experience at NASA Ames

At the NASA-Ames Research Center's Vertical Motion Simulator (VMS) facility nearly every simulated aircraft model is a custom development based on materials imported from elsewhere. Imported models have been received in the form of papers, notes, and computer files in nearly every conceivable manner. With today's emphasis on cost cutting there is a large interest in automating model importation into the Ames facilities since the work associated with implementing these models from these various source formats can take six staff-months or more. Consequently the DAVE-ML approach is welcomed because of its potential for use in automated code generation and verification.

Within the VMS facility the simulation model is programmed in the Fortran and Function Table Processor (FTP) languages. Other software languages may be used, such as C or ADA, but as the debugging environment is most compatible with Fortran and FTP, these are the favored modeling implementation languages. The Fortran programming language is no doubt familiar to the reader; FTP is a custom compiler developed at Ames that converts large, multidimensional data tables describing non-linear functions into Fortran-callable subroutines. Since DAVE-ML models normally include function tables, a means to convert from the XML statements of DAVE-ML into FTP input files was necessary. However, DAVE-ML includes more information than is typically necessary by the FTP language: DAVE-ML models may also include information about the origins of the data, statistical information, build-up equations, and checkcase data.

Two programs have been developed in the perl scripting language to automate conversion to and from the DAVE-ML XML format. *Elements.pl* converts a model from DAVE-ML format to FTP and Fortran code. *Elements.pl* is compatible with DAVE-ML version 1.53 and has been used to convert the generic fighter and HL-20 DAVE-ML aerodynamic models. A second program, *ftp2dave.pl*, converts the tabular functions in an Ames' FTP file into the DAVE-ML format.

The difference in feature sets of the various languages creates an interesting situation. Since the FTP standard feature set constitutes only a subset of that of DAVE-ML (for example, FTP does not know or care about the source, or provenance, of the data), inclusion of items such as the provenance of the functions cannot be automated because the provenance is not included in a machine-readable way in an FTP input file (except possibly as an unstructured comment.) However an XML output file generated from an FTP file will properly contain the essentials of the tabular functions.

In addition, a suite of programs have been developed to run static check cases included in models compatible with DAVE-ML version 1.7.

A. Converting DAVE-ML Models with Perl

All Ames' programs that convert to and from DAVE-ML have been written in perl. Perl is a general-purpose programming language that was originally developed for text manipulation and now is used for a wide range of tasks including system administration, web development, and network programming. Since perl is open source software, it is available for many operating system platforms including Windows, Macintosh®, Unix and VMS®. Since a DAVE-ML aerodynamics file is all UNICODE text (which contains a larger set of characters than ASCII), perl's UNICODE text handling capabilities are especially well suited to convert the XML form to other text forms such as the Fortran and FTP code files.

The publicly-available "XML::Xpath" perl module enables the *elements.pl* program to parse a DAVE-ML model. This module is based on the XPath specification available from the World Wide Web Consortium[*]. The specification provides a common syntax, which may be used to address parts of an XML document and provides many basic facilities for manipulation of the document content. XPath gets its name from its use of a path notation

[*] http://www.w3.org/TR/xpath [cited July 5, 2004]

similar to that of Uniform Resource Locators (URLs) for navigating through the hierarchical structure of an XML document. An example path, appropriate to a DAVE-ML document, would be the path 'DAVEfunc/function', which is used to access all the instances of the element `<function>` in a DAVE-ML model. This faculty greatly facilitates access to the portions of the XML code of interest. The program module reads an entire XML document into memory and models it as a tree of nodes for quick reference later. The nodes of interest are element nodes, attribute nodes and text nodes of the DAVEfunc specification. XML::XPath provides functions for accessing them all. [In perl, program modules perform the function of specialized libraries. Dozens if not hundreds of program modules are freely available from the Comprehensive Perl Archive Network (CPAN)[*] and are easily loaded and configured onto a computer. There are more than a dozen available for dealing with XML alone. XPath was selected for its simplicity of use.]

Another perl module, "XML::Writer," is used in *ftp2dave.pl* as a helper program module to write XML documents. The module handles all escaping for attribute values and character data and constructs different types of markup, such as the node tags, attributes, text, comments, and processing instructions. By default, the module performs several well-formedness checks to catch errors during output.

B. Utility for Converting DAVE-ML Models to NASA Ames format (Elements.pl)

The *elements.pl* script satisfactorily processes a DAVE-ML-based XML file and produces FTP and Fortran source code. Two aspects of the *elements.pl* program are worthy of comment. These are its handling of mathematical equation representations in DAVE-ML and its handling of functions which share data tables (included in DAVE-ML version 1.5) to simplify and shorten the DAVE-ML file.

1. Math Processing

Elements.pl's processing roughly follows the DAVE-ML Document Type Definition[†] (DTD) order. Most of the early portions of a model file are provenance, modification histories, variable definitions and the like. Most of this information is immediately output as FTP comments. However, some of the variable definitions consist of equation build-up formatted in the MathML markup language and is output to a Fortran code fragment. Handling these formulations required parsing logic to recognize and translate the MathML elements such as `<divide>`, `<plus>`, and `<times>` into Fortran renditions. An extensive list of mathematical operators (e.g. + – () abs * /) was included along with their Fortran precedence to facilitate parsing. The list of the intrinsic built-in functions forms a sub-set of the Fortran specification but it is extensive enough to handle aerodynamic formulations. Naturally, equations can extend over many lines of code; *elements.pl* formats the line extensions correctly in Fortran.

2. Shared Data Tables

With the advent of version 1.51b of the DTD the capability to share function table became available. It is a logical convenience to be able to define a large data table once and use it multiple times. Such occurrences are common with aircraft having identical table definitions of a variable such as Lift being identical for left and right flaps. In version 1.51b the XML code uses a `<griddedTableDef>` element to contain the table data as well as other information about the data and then subsequently refers to the data set through a `<griddedTableRef>` element when the table is needed for definition of a specific functional relationship. This is a good idea that conserves space and facilitates visual inspection of the code listings. Some aerodynamic models contain dozens of such functions making the efficiency welcome.

The FTP language supports the reuse of tabular data and *elements.pl* support this concept. The FTP keyword `EQUIV` allows one to reference a table that already was defined in an FTP language `POINTS` statement. When *elements.pl* is formulating its FTP output, the first time the data should be output, it is output as a `POINTS` statement. Thereafter, the table is reused via an FTP `EQUIV` statement.

C. Fortran Code Sorting Utility (Sortfort.pl)

The resulting Fortran code from *elements.pl* is not guaranteed to be sequenced correctly; it usually needs to be sorted into the required logical computation precedence before being compiled. This is a consequence of the ordering of the elements in the DAVE-ML DTD.

There are six classes of Fortran statements that can result from processing a model file with *elements.pl*. These are comments, equations, calls, Fortran functions, `IF` blocks and `DATA` statements. At present the comments are

[*] http://www.cpan.org/ [cited July 5, 2004]

[†] available from http://dcb.larc.nasa.gov/utils/fltsim/DAVE/DTDs.html [cited July 5, 2004]

discarded, the `END` statement is replaced with the Fortran `STOP` and `END` statements and the rest are sorted into the appropriate order. Retaining the comments is a desirable enhancement to be made in future versions.

Making use of perl's excellent data structure capability, each of the Fortran statements (or blocks) is examined for "inputs" and "outputs." These are saved away in an associative array along with the actual Fortran statement and a flag indicating whether it has been written to the output Fortran file. As an example, inputs and outputs for an equation are fairly obvious; they are the independent variables and the dependent variable. The `IF` block requires detailed attention as it may have inputs in its logical expression, in the equations of its alternatives, and may contain multiple outputs in its alternative sections.

The basic idea of the sorting algorithm is to recursively examine each output variable to see if its inputs have been defined as an output elsewhere in the associative array. If so, its inputs are examined recursively and so on until there is no such output for the last input. At that point the last output expression can be written to the Fortran file and is marked as having been written. The process continues examination of the associative array elements until all output variables in the array have been examined.

Input variables that remain are noted in a Fortran comment as "probably input variables" to facilitate subsequent checking or integration into a simulation model containing all the subsystems. The table names generated by the FTP compiler and used by its runtime modules are grouped together and output as Fortran `EXTERNAL` statements. As an example of how large some of the aerodynamic models are, the file output of *hl20.for* lists over 150 `EXTERNAL` statements.

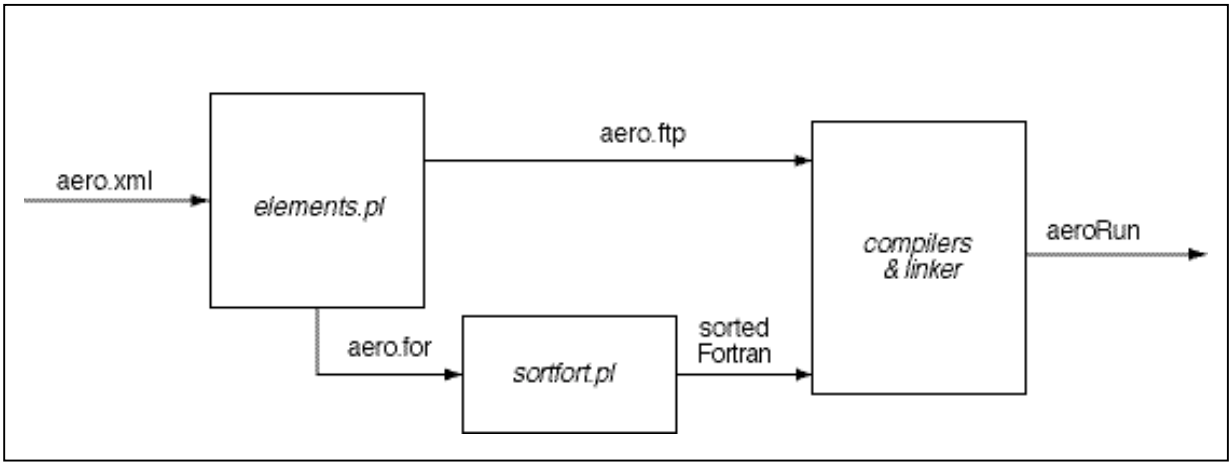
D. Utility for Converting a NASA Ames Function to DAVE-ML (Ftp2dave.pl)

This program performs the basic conversion of an FTP source code aerodynamic tabular model into a DAVE-ML-based XML files. While it performs correctly, it is only compliant with version 1.4 of the DAVEfunc DTD. It outputs duplicates of the data tables if the tables are used more than once. Compliance with version 1.5 would be more efficient in this regard.

Many of the useful documentary elements in DAVE-ML have no counterpart in the FTP language. Thus the provenance of the model and tables, references and the modification records of the file header element are not provided under FTP. These are very valuable pieces of information that really should be included in the XML representation. Enhancing *ftp2dave.pl* by adding these documentary features and bringing the program up to version 1.5 would be of great value.

IV. Implementation Results/Status at Ames

The *elements.pl* script satisfactorily processes a DAVE-ML-based XML file and produces FTP and Fortran source code files. The FTP file is complete and may be compiled and linked in with other parts of a model. The Fortran file may be used to create the aerodynamic subroutine. The file is actually a Fortran "Main" file in that it can be compiled and linked and subsequently executed as in Fig. 1. Of course, there are no output statements so it produces no useful output in this mode.



With the introduction of DAVE-ML version 1.7, static check data is defined as part of the DAVEfunc DTD and is available for the HL-20 aerodynamic model. Having check data provides a means to verifying that the conversion routines work correctly and can actually produce something worthwhile. The program *genck.pl* was devised to

generate Fortran code files for inserting input check data and output print statements for performing one of the included checks. These files are code fragments, which must be combined with the sorted Fortran code. This is accomplished with *combine.pl*. A "make" file allows one to specify which test is desired and subsequently generates all the code, combines it and outputs an executable program for checking that one case. Figure 2 illustrates the process. While the intent of providing check cases is to verify ultimate simulation model transfer correctness, they have been invaluable in debugging the conversion programs.

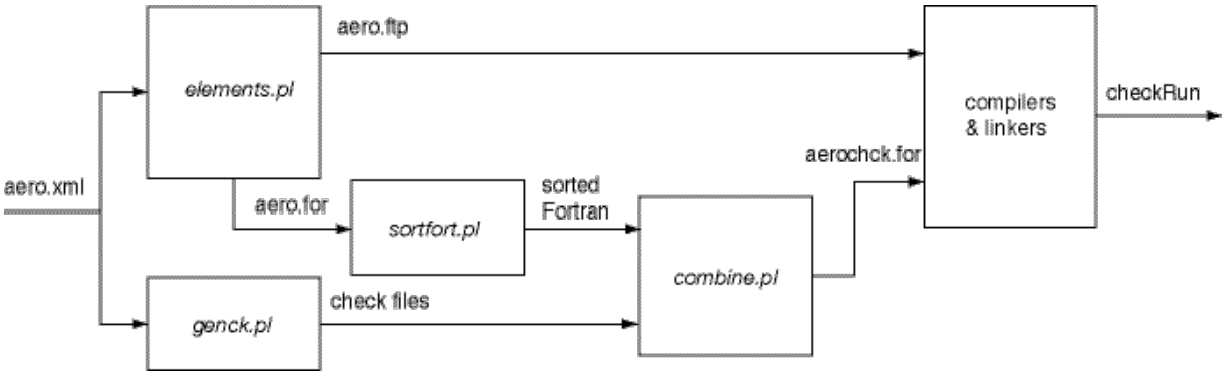


Figure 2. "Making" the check executable

Reflecting on the experience of the conversion of DAVE-ML code to the Ames paradigm (Fortran and FTP), one could expect other facilities to experience similar challenges. It is fairly straightforward to decide what must be done to convert a DAVE-ML aero model to and from Fortran and FTP. Some technological problems centered on Fortran such as handling initial values of variables, conversion of MATHML statements, and placing Fortran statements in correct computational order. Problems with FTP stemmed from the fact that FTP contains a subset of the required elements of DAVE-ML. This makes automated conversion from FTP inadequate because it cannot provide valid DAVE-ML code (valid in the XML sense.) Consequently human editing becomes necessary. Another challenge consists in combining the DAVE-ML code from the twin output streams from Fortran and FTP source. None of these problems was insurmountable. It is to be expected that simulation facilities will find the Ames experience to be more or less their own. Based on the experience of converting the two example models (generic fighter and HL-20), it can be said that the Ames tools and utilities are such that given a new DAVE-ML aero model, the Ames facility could convert it to Ames' preferred Fortran and FTP form in about a day and could integrate it into a full model in about a week. Converting from an Ames model expressed in Fortran and FTP to DAVE-ML is untried at this point but is expected to attain a similar efficiency. This is a very satisfactory situation and bodes well for the future of DAVE-ML.

V. Implementation experience at NAVAIR Manned Flight Simulator

At the Manned Flight Simulator (MFS) on NAS Patuxent River, Maryland, all simulations used for research, analysis, and training, both at the desktop and with a pilot in the loop, are hosted under a standard environment called CASTLE[8]. Historically, when a new simulation has been brought into the facility from an outside source, a great deal of labor has been required to prepare the source code and data for execution under CASTLE. For some simulations, improvements to the fidelity of the model are ongoing by outside sources, and the MFS implementation must be continually updated using new datasets. Typically, each new dataset must go through a laborious process of conversion to the CASTLE format and subsequent checkout. Although many programs have automated this effort, each separate simulation project has its own methodology for the process. There is no standard mechanism whereby external simulation models can be ported to CASTLE and subsequently maintained.

At the same time the DAVE-ML standard was initially being created, a new version of CASTLE was being designed. This new version, called CASTLE v6.0, or "Citadel," incorporates many new, modern features, such as a robust, user-friendly interface, complete automation through an easy-to-learn scripting language, and the use of XML for all simulation data files. Since DAVE-ML is in fact XML, and Citadel already includes the facilities for

reading and writing XML source, it was a natural progression to the inclusion of code within Citadel to read a DAVE-ML model natively. For the CASTLE solution, no intermediate translation steps are required; Citadel recognizes either the existing CASTLE v5.0 data format or the new (and now preferred) standard DAVE-ML data format.

As a test of the nascent DAVE-ML support within Citadel, the example aerodynamics model supplied by NASA Langley was loaded directly into CASTLE, and simple checkcases were run to ensure that the model was read correctly. It was shown that a complete non-linear, time-independent aerodynamics model, including build-up equations, could be described using DAVE-ML, loaded into CASTLE, and used to generate data. In such a case, there is no compiled source code associated with the aerodynamics model; instead, the model is completely defined by the DAVE-ML file that has been designated as part of the simulation. While it has not yet been literally done, a model converted by NASA Ames from their native format to DAVE-ML using the tools described above could be immediately loaded into CASTLE and used for analysis at MFS. A future goal of this work is to demonstrate just such an operation, showing that the entire conversion process would take hours or days instead of months.

Work with DAVE-ML at NAVAIR is ongoing. The latest version of the DAVE-ML standard contains the definition of checkcase data within the DAVE-ML file. In the near future, CASTLE will be expanded to include an automatic checkout capability, whereby the user of a simulation containing a DAVE-ML data file can simply select a "Verify Data" menu option, and CASTLE will automatically run the checkcases present within the file. Upon completion of the tests, CASTLE will be able either to provide a "go/no-go" result from the tests, or to generate a complete set of verification plots if required.

In addition to the above work, a simple generic high-performance fighter aircraft aerodynamics model used for CASTLE demonstrations is being converted to DAVE-ML through a combination of manual and automatic processes. This model will be passed to NASA Ames for import into their simulation environment as further validation and demonstration of the utility of DAVE-ML for data interchange between disparate facilities.

CASTLE v6.0 is slated for release in early 2005, including the ability to read DAVE-ML directly. It is hoped that the adoption and preference of DAVE-ML format for aero models by CASTLE will aid in the acceptance of this emerging AIAA standard.

VI. Ongoing DAVE-ML development efforts

In parallel with these demonstration efforts at NASA Ames and NAVAIR Patuxent River, further work on refining the DAVE-ML format has been underway. Suggestions from the demonstration participants led to improvements in the structure and capabilities of DAVE-ML, especially in regard to verification test data and the publication of an on-line reference manual[*]. Uncertainty statistics can now be included in the model to support Monte Carlo analyses using these models.

Work continues on refining editing and analysis tools that support DAVE-ML, including

- A DAVE-ML to Simulink® translator[†], written in Java for portability, that constructs Simulink® block diagrams from a DAVE-ML file. This tool (*dave2sl*) has been used (internally by NASA) to support several flight projects, including the second X-43 Hyper-X flight, the NASA/Boeing X-48 Blended-Wing-Body subscale research aircraft and analysis of Boeing's X-37 Atmospheric Launch Test Vehicle.
- A DAVE-ML to XHTML converter, written using eXtensible Stylesheet Language (XSL)[‡], that creates documentation directly from a version 1.4 DAVE-ML model to a XHTML web document (*DAVE_html.xsl*).
- A command line model exerciser, written in Java for portability, that calculates static outputs based on static input values for a DAVE-ML model (*dave*).

A mailing list to discuss simulation model standardization issues and to announce upcoming events was been established; interested individuals can subscribe by following instructions found on the DAVE-ML project web site[§].

VII. Concluding remarks

Based on results obtained thus far, the proposed AIAA standard as realized in DAVE-ML has been demonstrated to provide a viable means to convey the aerodynamics portion of a flight dynamics model from one facility to

[*] http://dcb.larc.nasa.gov/utils/fltsim/DAVE/DTDs.html [cited July 5, 2004]
[†] http://dcb.larc.nasa.gov/utils/fltsim/DAVE/DAVEtools.html [cited July 5, 2004]
[‡] http://www.w3c.org/TR/xsl [cited July 5, 2004]
[§] http://dcb.larc.nasa.gov/utils/fltsim/DAVE/index.html [cited July 5, 2004]

another. Scripts to convert from DAVE-ML into Fortran, C++ and Simulink® with verification have been developed at three simulation facilities, thus automating a previously manual process. One participant has developed rudimentary Fortran/FTP to DAVE-ML export scripts as well. What remains to be demonstrated is a complete end-to-end demonstration – that is, taking a aerodynamics model from its native implementation at a facility and expressing it in DAVE-ML, and successfully importing the model at another facility with little or no human interaction.

NASA Ames has successfully developed import routines that convert aerodynamic models and their associated function tables from DAVE-ML to NASA Ames Fortran and function tables. This has been estimated to reduce the time to import a model to Ames from several weeks to no more than one week. Ames has also developed export routines that converts Ames function tables to the DAVE-ML format.

NAVAIR is well along in the development of their latest simulation architecture (CASTLE 6.0/Citadel) that accepts DAVE-ML models directly into the simulation in their native form, eliminating the need for explicit import and export.

Additional development efforts should continue to extend DAVE-ML to cover more data types essential to fully automate the transfer of a complete flight dynamics model between simulation facilities; this would help improve the efficiency of aerospace development teams, reduce the burden of government in oversight of contracted development, and lower the cost of maintaining deployed training devices.

Appendix

The following is excerpted from the generic fighter subsonic aerodynamics DAVE-ML pathfinder model used in this demonstration. The complete version is available at the DAVE-ML project web site[*].

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE DAVEfunc SYSTEM "DAVEfunc.dtd">
<!-- $Revision: 2.3 $ -->
<DAVEfunc>
    <fileHeader name="F-16 Subsonic Aerodynamics Model (a la Garza)" >
        <author name="Bruce Jackson" org="NASA Langley Research Center" xns="@bjax"/>
        <fileCreationDate date="10-JUNE-2003"/>
        <description>
           F-16 Aero Data file. Based on Morelli's adaptation of
          Stevens and Lewis' F-16 example [1] described in Garza &amp;amp;
          Morelli's TM [2]. Obtained from E. A. Morelli in the form of
          Matlab scripts [3] &amp;amp; [4]. This version has quotient's
          replaced with divide's.
        </description>

      <!-- ================== -->   [Comments are included to make the file more human-readable]
      <!--     References     -->
      <!-- ================== -->

      <reference refID="REF01" author="Stevens, Brian L. and Lewis, Frank L."
         title="Aircraft Control and Simulation"
         accession="ISBN O-471-61397-5" date="1992"/>

      <reference xlink:href=
        "http://techreports.larc.nasa.gov/ltrs/PDF/2003/tm/NASA-2003-tm212145.pdf"
        refID="REF02" author="Garza, Fredrico R.; and Morelli, Eugene A."
        title="A Collection of Nonlinear Aircraft Simulations in MATLAB"
        accession="NASA TM-2003-212145" date="JAN-2003"/>

      <reference refID="REF03" author="Morelli, Eugene A."
         title="f16_aero.m" date="17-JUN-1995"/>

      <reference refID="REF04" author="Morelli, Eugene A."
         title="f16_aero_setup.m" date="17-JUN-1995"/>

      <reference refID="NOTE1"
        author="Bruce Jackson"
        title="Checkcase data source"
        date="2004-02-13">

        <description>
          From E. A. Morelli's f16 matlab script, dated 09-Oct-2000;
          Generated using 'make_DAVEML_checkcases.m'
        </description>

      </reference>

      <modificationRecord modID="A">
        <author name="Bruce Jackson" org="NASA Langley Research Center"
           email="e.b.jackson@nasa.gov"/>
        <description>
          Added checkcase static shots and internal values for debugging purposes.
        </description>
      </modificationRecord>

    </fileHeader>
```

[*] http://dcb.larc.nasa.gov/utils/fltsim/DAVE [cited July 5, 2004]

```
    <!-- ================== -->   [All input signals are defined; these should follow draft AIAA naming
    <!--  Input variables   -->   conventions.]
    <!-- ================== -->

  <variableDef name ="True_Airspeed_f_p_s" varID="vt" units= "ft/sec" symbol="Vt">
    <description>
      True airspeed, ft/sec
    </description>
  </variableDef>

  <variableDef name="Angle_of_Attack_deg" varID="alpha" units="deg" symbol="&#x3B1;">
    <description>
      Instantaneous true angle-of-attack, in degrees
    </description>
  </variableDef>

  <variableDef name="s_Body_Pitch_Rate_rad_p_s" varID="q" units="rad/sec"
               sign="aircraft nose up" symbol="q">
    <description>
      Body pitch rate, rad/sec
    </description>
  </variableDef>
```

.
.
.

[several input variables have been excised to save space]

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

  <variableDef name="rtd" varID="rtd" units="deg/rad">
    <description>
   Conversion constant from radians to degrees
    </description>
    <calculation>              [An example of MathML, here used to specify the value of the constant
      <math>                   rtd = 180/π]
        <apply>
          <divide/>
          <cn>180.</cn>
          <cn>3.14159265</cn>
        </apply>
      </math>
    </calculation>
  </variableDef>

  <variableDef name="cbar" varID="cbar" units="ft" initialValue="11.32">
    <description>
       Length of aerodynamic chord, ft
    </description>
  </variableDef>               [Constants may be specified by a simple numeric initial value]
```

.
.
.

[several constants have been excised to save space]

```
    <!-- ================== -->
    <!--  Local variables   -->
    <!-- ================== -->
```
[In a `variableDef`, a `name` attribute is used for documentation. The `varID` attribute is used by the import tool in processing the model

```
<variableDef name="del" varID="del" units="">
  <description>
 Normalized elevator deflection (note range is beyond breakpoint range)
  </description>
  <calculation>
    <math>
 <apply>
   <divide/>
   <ci>el</ci>
   <cn>25.0</cn>
 </apply>
    </math>
  </calculation>
</variableDef>
```

.
.
.

[several local variables have been excised to save space]

```
<!-- Aerodynamic Forces -->

<variableDef name="CX0" varID="cxt" units="" sign="forward">
  <description>
    Basic coefficient of force in the X-body direction
    (+FWD). Function table based on angle of attack and elevator
    deflection.
  </description>
</variableDef>

<variableDef name="CY0" varID="cy0" units="" sign="right">
  <description>
 Basic coefficient of force in the Y-body direction
 (+right). Function of sideslip angle, aileron and rudder
 deflection.
  </description>
  <calculation>
    <math>
      <apply>
        <plus/>
        <apply>
          <times/>
          <cn>-0.02</cn>
          <ci>beta</ci>
        </apply>
        <apply>
          <times/>
          <cn>0.021</cn>
          <ci>dail</ci>
         </apply>
         <apply>
          <times/>
          <cn>0.086</cn>
          <ci>drdr</ci>
         </apply>
      </apply>
    </math>
  </calculation>
</variableDef>
```

[This shows how a variable's calculation is defined in MathML syntax based on other variables (`<ci>`) or numeric values (`<cn>`). Here we are specifying that `cy0 = -0.02*beta + 0.021*dail + 0.086*drdr`].

```
  <variableDef name="CZ0" varID="czt" units="" sign="down">
    <description>
      Basic coefficient of force in the Z-body direction
      (+DWN). Function table based on angle of attack and elevator
      deflection.
    </description>
  </variableDef>
.
.
.
```

[several aerodynamic force variable definitions have been excised to save space]

```
  <!-- Function table lookup support variables -->

  <variableDef name="absbeta" varID="absbeta" units="deg">
    <description>
       Absolute value of angle-of-sideslip, deg.
    </description>
    <calculation>
      <math>
        <apply>
          <abs/>
          <ci>beta</ci>
        </apply>
      </math>
    </calculation>
  </variableDef>
```

[An example of specifying the `abs()` function]

```
.
.
.
```

[several elements have been excised to save space]

```
  <variableDef name="Cl0" varID="clt" units="" sign="right wing down">
    <description>
      Basic coefficient of moment around the X-body direction (roll) (+RWD)
    </description>
    <calculation>
      <math>
        <apply>
          <piecewise>
            <piece>
              <apply>
                <minus/>
                <ci>absCl0</ci>
              </apply>
              <apply>
                <lt/>
                <ci>beta</ci>
                <cn>0</cn>
              </apply>
            </piece>
            <otherwise>
              <ci>absCl0</ci>
            </otherwise>
          </piecewise>
        </apply>
      </math>
    </calculation>
  </variableDef>
```

A more involved calculation implementing an if-then-else construct:

$$clt = \begin{cases} -\text{absCl0} & \text{if } \beta < 0 \\ \text{absCl0} & \text{otherwise} \end{cases}$$

```
<!-- Damping derivative tables -->

  <variableDef name="Cmq" varID="cmq" units="per rad" sign="up/+up">
    <description>
      Damping derivative: pitching moment increment per radian/sec of pitch rate
    </description>
  </variableDef>

  <variableDef name="Cnr" varID="cnr" units="per rad" sign="right/+right">
    <description>
      Damping derivative: yawing moment increment per radian/sec of yaw rate
    </description>
  </variableDef>

  <variableDef name="Cnp" varID="cnp" units="per rad" sign="right/+right">
    <description>
      Damping derivative: yawing moment increment per radian/sec of roll rate
    </description>
  </variableDef>
.
.
.
```

[several damping derivatives have been excised to save space]

```
      <!-- ================== -->
      <!--  Output variables  -->          [Final buildup equations]
      <!-- ================== -->

  <variableDef name="CX" varID="cx" units="" sign="FWD">
    <description>
      Total coefficient of force along the body X-axis.
    </description>
    <calculation>
      <math>
        <apply>
          <plus/>
          <ci>cxt</ci>
          <apply>
            <times/>
            <ci>cq2v</ci>
            <ci>cxq</ci>
          </apply>
        </apply>
      </math>
    </calculation>
</variableDef>

  <variableDef name="CZ" varID="cz" units="" sign="DOWN">
    <description>
      Total coefficient of force along the body Z-axis.
    </description>
    <calculation>
      <math>
        <apply>
          <plus/>
          <ci>cz1</ci>
          <apply>
            <times/>
            <ci>cq2v</ci>
            <ci>czq</ci>
          </apply>
        </apply>
      </math>
    </calculation>
```

```
</variableDef>
  <variableDef name="CLM" varID="cm" units="" sign="ANU">
    <description>
      Total coefficient of moment around the body Y-axis (pitching moment)
    </description>
    <calculation>
      <math>
        <apply>
          <plus/>
          <ci>cmt</ci>
          <apply>
            <times/>
            <ci>cq2v</ci>
            <ci>cmq</ci>
          </apply>
          <apply>
            <times/>
            <ci>cz</ci>
            <apply>
              <plus/>
              <ci>xcgr</ci>
              <apply>
                <minus/>
                <ci>xcg</ci>
              </apply>
            </apply>
          </apply>
        </apply>
      </math>
    </calculation>
  </variableDef>

.
.
.
```

[several elements have been excised to save space]

```
      <!-- ================== -->
      <!--  Breakpoint values -->          [Breakpoint sets can be shared by multiple tables.]
      <!-- ================== -->

  <breakpointDef name="alpha" bpID="ALPHA1" units="deg">
      <description>
    Alpha breakpoints for basic and damping aero tables
      </description>
      <bpVals>
   -10., -5., 0., 5., 10., 15., 20., 25., 30., 35., 40., 45.
      </bpVals>
  </breakpointDef>

  <breakpointDef name="beta" bpID="BETA1" units="deg">
    <description>
       Angle-of-sideslip breakpoints for basic aero tables
    </description>
    <bpVals>
        0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0
    </bpVals>
  </breakpointDef>

  <breakpointDef name="beta" bpID="BETA2" units="deg">
    <description>
       Angle-of-sideslip breakpoints for control power tables
    </description>
```

```
  <bpVals>
      -30.0, -20.0, -10.0, 0.0, 10.0, 20.0, 30.0
  </bpVals>
</breakpointDef>

<breakpointDef name="el" bpID="DE1" units="deg">
  <description>
    Elevator angle breakpoints
  </description>
  <bpVals>
    -24., -12., 0., 12., 24.
  </bpVals>
</breakpointDef>

    <!-- ================== -->
    <!--  Basic Functions   -->
    <!-- ================== -->

<function name="Basic CX">          [An example two-dimensional function: cxt = f(el, alpha)]
  <description>
    Basic coefficient of X-force table as a function of angle of attack and elevator
  </description>
  <provenance>
    <author name="Bruce Jackson" org="NASA Langley Research Center" xns="@bjax"/>
    <functionCreationDate date="28-MAR-2002"/>
    <documentRef docID="REF01"/>
    <documentRef docID="REF02"/>
    <documentRef docID="REF03"/>
  </provenance>
  <independentVarRef varID="el"    min="-24.0" max="24.0" extrapolate="neither"/>
                                                              <!-- DE breakpoints -->
  <independentVarRef varID="alpha" min="-10.0" max="45.0" extrapolate="neither"/>
                                                             <!-- Alpha breakpoints -->
  <dependentVarRef varID="cxt"/>

  <functionDefn name="CX_fn">
    <griddedTable name="CX_table">
      <breakpointRefs>
        <bpRef bpID="DE1"/>
        <bpRef bpID="ALPHA1"/>
      </breakpointRefs>
      <dataTable> <!-- Note: last breakpoint changes most rapidly -->
   -.099,-.081,-.081,-.063,-.025,.044,.097,.113,.145,.167,.174,.166,
   -.048,-.038,-.040,-.021,.016,.083,.127,.137,.162,.177,.179,.167,
   -.022,-.020,-.021,-.004,.032,.094,.128,.130,.154,.161,.155,.138,
   -.040,-.038,-.039,-.025,.006,.062,.087,.085,.100,.110,.104,.091,
   -.083,-.073,-.076,-.072,-.046,.012,.024,.025,.043,.053,.047,.040
      </dataTable>
    </griddedTable>
  </functionDefn>
</function>
```

```
  <function name="Basic CZ">        [A one-dimensional function:  czt = f(alpha)]
    <description>
      Basic coefficient of Z-force table as a function of angle of attack
    </description>
    <provenance>
      <author name="Bruce Jackson" org="NASA Langley Research Center" xns="@bjax"/>
      <functionCreationDate date="28-MAR-2002"/>
      <documentRef docID="REF01"/>
      <documentRef docID="REF02"/>
      <documentRef docID="REF03"/>
    </provenance>
    <independentVarRef varID="alpha" min="-10.0" max="45.0" extrapolate="neither"/>
                                                          <!-- Alpha breakpoints -->
    <dependentVarRef varID="czt"/>

    <functionDefn name="CZ0_fn">
      <griddedTable name="CZ0_table">
        <breakpointRefs>
          <bpRef bpID="ALPHA1"/>
        </breakpointRefs>
        <dataTable>
          .770,.241,-.100,-.416,-.731,-1.053, -1.366,-1.646,-1.917,-2.120,-2.248,2.229
        </dataTable>
      </griddedTable>
    </functionDefn>
  </function>

  <function name="Basic Cm">
    <description>
      Basic coefficient of pitching-moment as a function of angle of attack and
      elevator
    </description>
    <provenance>
      <author name="Bruce Jackson" org="NASA Langley Research Center" xns="@bjax"/>
      <functionCreationDate date="28-MAR-2002"/>
      <documentRef docID="REF01"/>
      <documentRef docID="REF02"/>
      <documentRef docID="REF03"/>
    </provenance>
    <independentVarRef varID="el"    min="-24.0" max="24.0" extrapolate="neither"/>
                                                            <!-- DE breakpoints -->
    <independentVarRef varID="alpha" min="-10.0" max="45.0" extrapolate="neither"/>
                                                          <!-- Alpha breakpoints -->
    <dependentVarRef varID="cmt"/>

    <functionDefn name="Cm0_fn">
      <griddedTable name="Cm0_table">
        <breakpointRefs>
          <bpRef bpID="DE1"/>
          <bpRef bpID="ALPHA1"/>
        </breakpointRefs>
        <dataTable> <!-- Note: last breakpoint changes most rapidly -->
   .205,.168,.186,.196,.213,.251,.245,.238,.252,.231,.198,.192,
   .081,.077,.107,.110,.110,.141,.127,.119,.133,.108,.081,.093,
   -.046,-.020,-.009,-.005,-.006,.010,.006,-.001,.014,.000,-.013,.032,
   -.174,-.145,-.121,-.127,-.129,-.102,-.097,-.113,-.087,-.084,-.069,-.006,
   -.259,-.202,-.184,-.193,-.199,-.150,-.160,-.167,-.104,-.076,-.041,-.005
        </dataTable>
      </griddedTable>
    </functionDefn>
  </function>
.
.
.
```

[several tables have been excised to save space]

```
    <!-- ================== -->
    <!-- Damping Derivatives -->
    <!-- ================== -->

    <!--  CXq CYr CYp CZq Clr Clp Cmq Cnr Cnp -->

<function name="CXq">
  <description>
    Damping derivative: axial force due to pitch rate as a function of angle-of-
    attack
  </description>
  <provenance>
    <author name="Bruce Jackson" org="NASA Langley Research Center" xns="@bjax"/>
    <functionCreationDate date="04-APR-2002"/>
    <documentRef docID="REF01"/>
    <documentRef docID="REF02"/>
    <documentRef docID="REF03"/>
  </provenance>
  <independentVarRef varID="alpha" min="-10.0" max="45.0" extrapolate="neither"/>
                                                        <!-- Alpha breakpoints -->
  <dependentVarRef varID="cxq"/>
  <functionDefn name="CXq_fn">
    <griddedTable name="CXq_table">
      <breakpointRefs>
        <bpRef bpID="ALPHA1"/>
      </breakpointRefs>
      <dataTable>
        -.267, -.110, .308, 1.34, 2.08, 2.91,  2.76,  2.05, 1.50, 1.49, 1.83, 1.21
      </dataTable>
    </griddedTable>
  </functionDefn>
</function>

<function name="CZq">
  <description>
    Damping derivative: normal force due to pitch rate as a function of angle-of-
    attack
  </description>
  <provenance>
    <author name="Bruce Jackson" org="NASA Langley Research Center" xns="@bjax"/>
    <functionCreationDate date="04-APR-2002"/>
    <documentRef docID="REF01"/>
    <documentRef docID="REF02"/>
    <documentRef docID="REF03"/>
  </provenance>
  <independentVarRef varID="alpha" min="-10.0" max="45.0" extrapolate="neither"/>
                                                         <!--Alpha breakpoints -->
  <dependentVarRef varID="czq"/>
  <functionDefn name="CZq_fn">
    <griddedTable name="CZq_table">
      <breakpointRefs>
        <bpRef bpID="ALPHA1"/>
      </breakpointRefs>
      <dataTable>
        -8.80, -25.8, -28.9, -31.4, -31.2, -30.7, -27.7, -28.2, -29.0, -29.8, -38.3,
        -35.3
      </dataTable>
    </griddedTable>
  </functionDefn>
</function>
```

```
  <function name="Cmq">
    <description>
      Damping derivative: pitching moment due to pitch rate as a function of angle-of-
      attack
    </description>
    <provenance>
      <author name="Bruce Jackson" org="NASA Langley Research Center" xns="@bjax"/>
      <functionCreationDate date="04-APR-2002"/>
      <documentRef docID="REF01"/>
      <documentRef docID="REF02"/>
      <documentRef docID="REF03"/>
    </provenance>
    <independentVarRef varID="alpha" min="-10.0" max="45.0" extrapolate="neither"/>
                                                          <!-- Alpha breakpoints -->
    <dependentVarRef varID="cmq"/>
    <functionDefn name="Cmq_fn">
      <griddedTable name="Cmq_table">
        <breakpointRefs>
          <bpRef bpID="ALPHA1"/>
        </breakpointRefs>
        <dataTable>
    -7.21, -5.40, -5.23, -5.26, -6.11, -6.64, -5.69, -6.00, -6.20, -6.40, -6.60, -6.00
        </dataTable>
      </griddedTable>
    </functionDefn>
  </function>
```

.
.
.

[several tables have been excised to save space]

```
      <!-- ================== -->
      <!--   Checkcase Data   -->
      <!-- ================== -->
```

[This section provides a set of verification test sets where inputs are specified…]

```
  <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>Angle_of_Sideslip_deg</signalName>
          <signalUnits>deg</signalUnits>
          <signalValue>   0.000</signalValue>
        </signal>
        <signal>
          <signalName>s_Body_Roll_Rate_rad_p_s</signalName>
          <signalUnits>rad/sec</signalUnits>
          <signalValue>   0.000</signalValue>
        </signal>
        <signal>
          <signalName>s_Body_Pitch_Rate_rad_p_s</signalName>
          <signalUnits>rad/sec</signalUnits>
          <signalValue>   0.000</signalValue>
```

```
      </signal>
      <signal>
        <signalName>s_Body_Yaw_Rate_rad_p_s</signalName>
        <signalUnits>rad/sec</signalUnits>
        <signalValue>   0.000</signalValue>
      </signal>
      <signal>
        <signalName>delta elevator</signalName>
        <signalUnits>deg</signalUnits>
        <signalValue>   0.000</signalValue>
      </signal>
      <signal>
        <signalName>delta aileron</signalName>
        <signalUnits>deg</signalUnits>
        <signalValue>   0.000</signalValue>
      </signal>
      <signal>
        <signalName>delta rudder</signalName>
        <signalUnits>deg</signalUnits>
        <signalValue>   0.000</signalValue>
      </signal>
      <signal>
        <signalName>Xcg</signalName>
        <signalUnits>fract</signalUnits>
        <signalValue>   0.250</signalValue>
      </signal>
    </checkInputs>

    <internalValues>                          [... and internal values are provided for debugging...]
      <signal>
        <signalID>vt</signalID>
        <signalValue>300.0</signalValue>
      </signal>
.
.
.
```

[several internal values have been excised to save space]

```
    </internalValues>

    <checkOutputs>                            [... and required output values within tolerance limits
      <signal>                                are specified for satisfactory verification after import.]
        <signalName>CX</signalName>
        <signalValue>-0.00400000000000</signalValue>
        <tol>0.000001</tol>
      </signal>
      <signal>
        <signalName>CY</signalName>
        <signalValue> 0.00000000000000</signalValue>
        <tol>0.000001</tol>
      </signal>
      <signal>
        <signalName>CZ</signalName>
        <signalValue>-0.41600000000000</signalValue>
        <tol>0.000001</tol>
      </signal>
      <signal>
        <signalName>CLL</signalName>
        <signalValue> 0.00000000000000</signalValue>
        <tol>0.000001</tol>
      </signal>
      <signal>
        <signalName>CLM</signalName>
        <signalValue>-0.04660000000000</signalValue>
```

```
          <tol>0.000001</tol>
        </signal>
        <signal>
          <signalName>CLN</signalName>
          <signalValue> 0.00000000000000</signalValue>
          <tol>0.000001</tol>
        </signal>
      </checkOutputs>
    </staticShot>

.
.
.
```

[other checkcases have been excised to save space]

```
  </checkData>
</DAVEfunc>
```

Acknowledgments

Simulink® is the registered trademark of a product of The Mathworks, Inc. of Natick, MA. Macintosh® is a registered trademark of Apple Computer, Inc. of Cupertino, CA. VMS® is a registered trademark of Hewlett-Packard Company of Palo Alto, CA.

The authors would like to thank Mr. Tom Alderete and Ms. Julie Mikula of NASA Ames Research Center, Dr. John B. Davidson and Dr. E. A. Morelli of NASA Langley Research Center, and Mr. Bill McNamara of NAVAIR Patuxent River and Mr. Jon Berndt, developer of JSBSim and contributor to the open-source FlightGear simulation project, for their support.

References

[1]AIAA Modeling and Simulation Technical Committee, "Standards for the Exchange of Simulation Modeling Data, Preliminary Draft," January 2003, URL: http://dcb.larc.nasa.gov/utils/fltsim/DAVE/AIAA_stds/SimDataExchange_Jan2003.pdf [cited July 5, 2004].

[2]Jackson, E. Bruce and Hildreth, Bruce L., "Flight Dynamic Model Exchange using XML," AIAA 2002-4482, August 2002.

[3]Yergeau, François, et al., "World Wide Web Consortium (W3C) Recommendation: XML 1.1," MIT, Cambridge MA, February 4, 2004, URL: http://www.w3c.org/TR/xml11 [cited July 5, 2004].

[4]Ausbrooks, Ron, et al., "World Wide Web Consortium (W3C) Recommendation: Mathematical Markup Language (MathML) Version 2.0 (Second Edition)," MIT, Cambridge MA, October 21, 2003, URL: http://www.w3c.org/TR/MathML2 [cited July 5, 2004].

[5]Nguyen, L. T., et al., "Simulator Study of Stall/Post-stall Characteristics of a Fighter Airplane with Relaxed Longitudinal Static Stability," NASA TP 1538, 1979, URL: http://techreports.larc.nasa.gov/ltrs/PDF/NASA-79-tp1538.pdf [cited July 5, 2004].

[6]Stevens, Brian L., and Lewis, Frank L., *Aircraft Control and Simulation*, Wiley & Sons, 1992, ISBN 0-471-61397.

[7]Garza, Fredrico R., and Morelli, Eugene A., "A Collection of Nonlinear Aircraft Simulations in MATLAB," NASA TM-2003-212145, January 2003, URL: http://techreports.larc.nasa.gov/ltrs/dublincore/2003/tm/NASA-2003-tm212145.html [cited July 5, 2004].

[8]York, Brent W., Magyar, Thomas J., and Nichols, James H. III, et al., "Citadel: the CASTLE v6.0 Project," AIAA 2002-4865, Monterey, CA August 2002.