



About Numerical Propulsion System Simulation (NPSS®)

Southwest Research Institute® (SwRI®), San Antonio TX

August 2020

Summary of Contents

Model Development.....	3
Problem Setup and Solution	5
Viewing Output Data	8
Standard NPSS Thermodynamic Packages.....	10
Standard NPSS Elements.....	11
NPSS Example Models	13
NPSS IDE.....	14
How to Get NPSS	15



NPSS is an object-oriented, multi-physics, engineering design and simulation environment which enables development, collaboration, and seamless integration of system models. Primary application areas for NPSS include aerospace systems (i.e. engine performance models for aircraft propulsion), thermodynamic system analysis such as Rankine and Brayton cycles, various rocket propulsion cycles, and industry standardization for model sharing and integration. However, since it is fundamentally a flow-network solver, it has also been applied to a variety of other fluid/thermal subjects such as multi-phase heat transfer systems, refrigeration cycles, variations of common power cycles (i.e. Brayton), and overall vehicle emission analyses.

Figure 1 shows a typical air breathing engine block diagram. The cycle shown in the block diagram is represented by the various thermodynamic processes such as compression, combustion, and turbine power extraction. When developing a model in the NPSS environment, the engineer specifies the type and order of the necessary components (referred to as “elements”) and provides the technical data which describes their individual performance. NPSS comes with a standard library of thermodynamic property databases and a library of standard elements to use in engine cycle models.

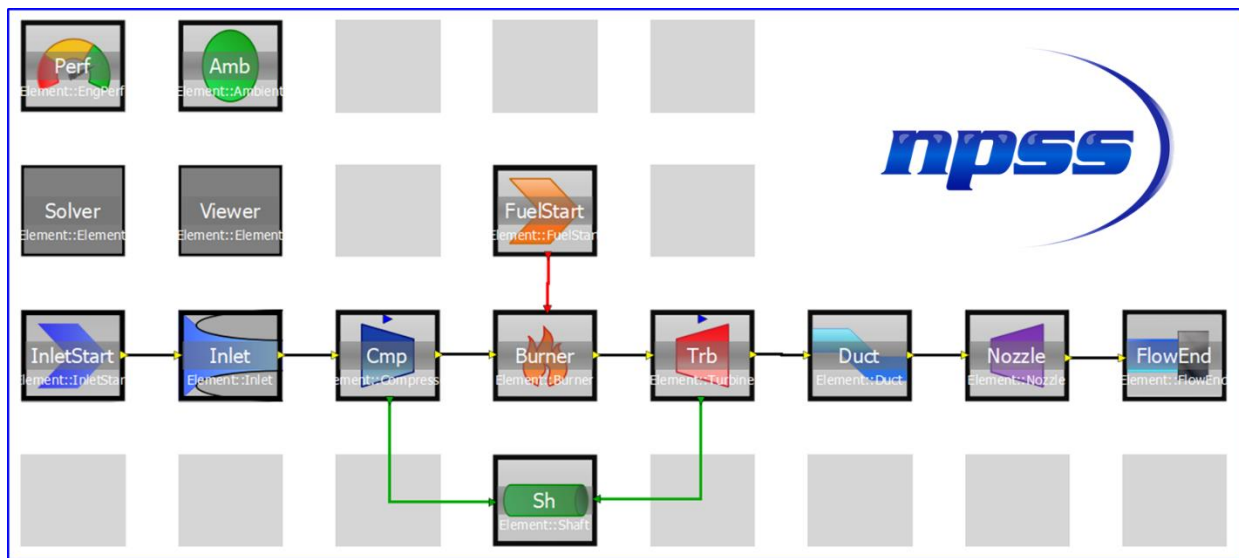


Figure 1. Typical Air Breathing Engine Block Diagram

Models can be defined interactively through the command window, but they are more commonly defined in standard text files. Simulations are typically launched in batch mode via a system command window, like the one shown on the far right of Figure 2. Most users select a text editor that supports language-specific highlighting, auto-complete, and version control features. Since NPSS is based on the C++ language, the C++ language setting in most text editors provides excellent readability of NPSS files. C++ syntax highlighting is used to view the model definition in the left and middle cascaded images of Figure 2.

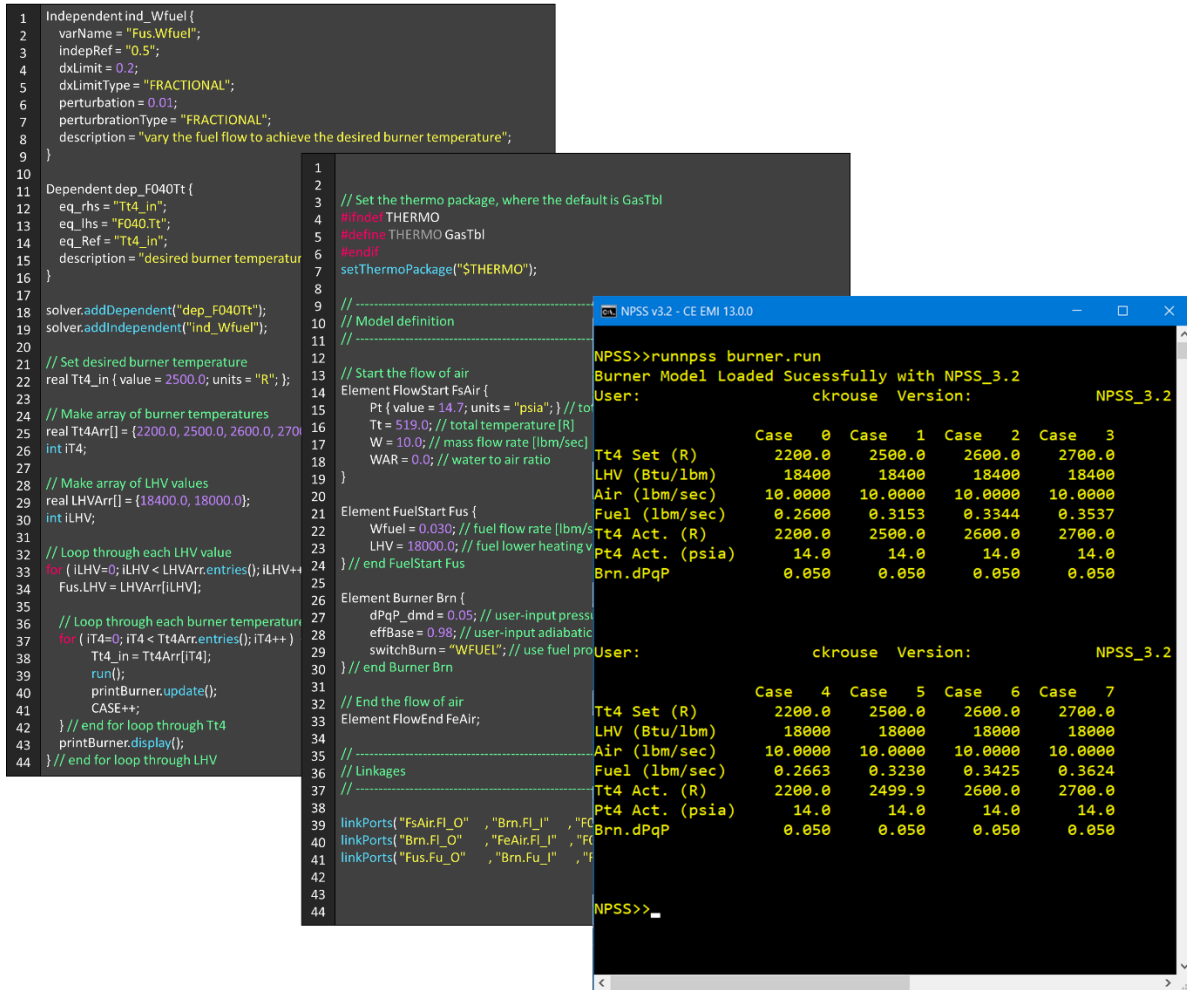


Figure 2. Left and Middle Images are NPSS Model Files that are Defined in a Standard Text Editor. Right Image is a System Command Window that is Typically Used to Run NPSS Models.

Model Development

Elements are specified and linked together in a user-defined input file, similar to the one shown in Figure 3. In this example, the “Cmp,” “Fus,” and “Brn” objects are based on the standard NPSS element classes, “Compressor,” “FuelStart,” and “Burner.” Within each of the objects, the engineer assigns the known physical parameters necessary for the problem solution, such as compressor pressure ratio, fuel flow rate, and burner efficiency. The various objects are then connected by using the NPSS linkPorts() function. For each link, the engineer inputs a meaningful name, such as “F020,” which signifies a connection at station number two of the air breathing engine cycle.

```

85 Element Compressor Cmp {
86     // set compressor design point
87     PRdes = 4.9; // pressure ratio
88     effDes = 0.85; // efficiency
89     Sh_O.inertia = 0.0005; // shaft inertia
90     // load file that instantiates a subelement for the compressor performance map
91     #include "hpcE3.map";
92 } // end Compressor Cmp
93
94 Element FuelStart Fus {
95     Wfuel = 0.030; // fuel flow rate [lbm/sec]
96     LHV = 18000.0; // fuel lower heating value [Btu/lbm]
97 } // end FuelStart Fus
98
99 Element Burner Brn {
100    dPqP_dmd = 0.05; // user-input pressure loss
101    effBase = 0.98; // user-input adiabatic efficiency
102    switchBurn = "WFUEL"; // use fuel properties from the FuelStart element
103 } // end Burner Brn
104
105
170 // -----
171 // fluid links
172 // -----
173
174 // Ambient to Inlet
175 linkPorts("InletStart.Fl_O", "Inlet.Fl_I", "F010" );
176
177 // Primary cold section
178 linkPorts("Inlet.Fl_O", "Cmp.Fl_I", "F020" );
179 linkPorts("Cmp.Fl_O", "Burner.Fl_I", "F030" );
180 linkPorts("FuelStart.Fu_O", "Burner.Fu_I", "FU036" );
181
182 // Primary hot section
183 linkPorts("Burner.Fl_O", "Trb.Fl_I", "F040" );
184 linkPorts("Trb.Fl_O", "Duct.Fl_I", "F070" );
185 linkPorts("Duct.Fl_O", "Nozzle.Fl_I", "F080" );
186 linkPorts("Nozzle.Fl_O", "FlowEnd.Fl_I", "F090" );
187
188
189

```

Figure 3. Example of Instantiating NPSS Objects (Left) and Linking Them Together (Right)

In addition to the standard library of elements provided by NPSS, the standard modeling environment enables the definition of customized elements. Via interpreted files, NPSS users can access the NPSS code used to define each element class, such as the “Compressor” class, and then they make their own modifications to the standard class and assign a new name, such as “CompressorSkywalker.” This is a powerful feature of NPSS that provides significant flexibility in the types of elements used in a model. Figure 4 shows an example of the standard NPSS “EngPerf” element on the left and the user-modified class code on the right. In this example, the user has modified the “EngPerf” class to calculate specific fuel consumption (SFC) based on shaft power and then convert the SFC units.

```

84 // -----
85 // Sum nozzle gross thrusts
86 // -----
87 Fg = 0.0;
88 for ( i = 0; i < _ptrFg.entries(); i++) {
89     Fg = Fg + _ptrFg[i]->value;
90 } // end for
91
92 // -----
93 // Calculations
94 // -----
95 Fn = Fg - Fram;
96
97 // if the static pressure string is empty, don't get its value
98 if ( _ptrPs != "" ) {
99     Fnc = Fn * C_PSTD / _ptrPs->value;
100 } // end if
101
102 Wfuel = Wfuel * 3600.0; // convert lbm/sec to lbm/hr
103 SFC = Wfuel / Fn;
104
105
106
107 } // end calculate()
108 } // end EngPerf class
109
170 // -----
171 // Sum torque loads
172 // -----
173 tPwr = 0.0;
174 for ( i = 0; i < _ptrLoad.entries(); i++) {
175     tPwr = tPwr + _ptrLoad[i]->value;
176 } // end for
177
178 // -----
179 // Calculations
180 // -----
181 Fn = Fg - Fram;
182
183 // if the static pressure string is empty, don't get its value
184 if ( _ptrPs != "" ) {
185     Fnc = Fn * C_PSTD / _ptrPs->value;
186 } // end if
187
188 Wfuel = Wfuel * 3600.0; // convert lbm/sec to lbm/hr
189 SFC = Wfuel / tPwr;
190 SFC_kW = SFC * 1.34; // convert to lb/kW/hr
191
192 } // end calculate()
193 } // end EndPerf class

```

Figure 4. Standard NPSS Code for the “EngPerf” Class (Left) and User-Modified “EngPerf” Class (Right)

Problem Setup and Solution

Once the model is developed, the engineer must then setup the problem and define the solution goals and constraints. The problem setup consists mainly of gathering all of the inputs required for a valid solution, defining the desired outputs and where to send them, and defining the solution cases to run. NPSS is organized using various types of input files. For large models and complicated systems, it is useful to organize the simulation using a variety of file types which are then linked together to form a complete simulation. From the SAE ARP5571 document, a typical simulation contains the types of files shown in Table 1. The recommended file directory structure for large models is shown in Figure 5.

However, there are no formal rules regarding the number, type, or organization of the input files. In the simplest models, the engineer can completely define the entire simulation in a single file.

File Extension	File Description
<i>.run</i>	Typically, the master file that defines the problem, includes the model, and specifies the thermodynamic package
<i>.case</i>	Typically, the file that contains data specific to a given case (e.g. design case or off-design case)
<i>.mdl</i>	Top-level model definition that contains all of the elements and connections required to represent the system
<i>.view</i>	Defines viewers that extract data from the system and format it into output files
<i>.map</i>	Defines the performance maps for components
<i>.inp / .inc</i>	Input or include files contain other information required for the overall simulation
<i>.fnc</i>	Contains user-defined functions for performing custom operations
<i>.enc</i>	Encrypted model
<i>.int</i>	Interpreted class definition that contains all of the engineering methods in a human-readable format (e.g. Compressor.int and EngPerf.int)
<i>.dll / .sc</i>	Compiled object that contains engineering methods

Table 1. Typical File Extensions and Descriptions

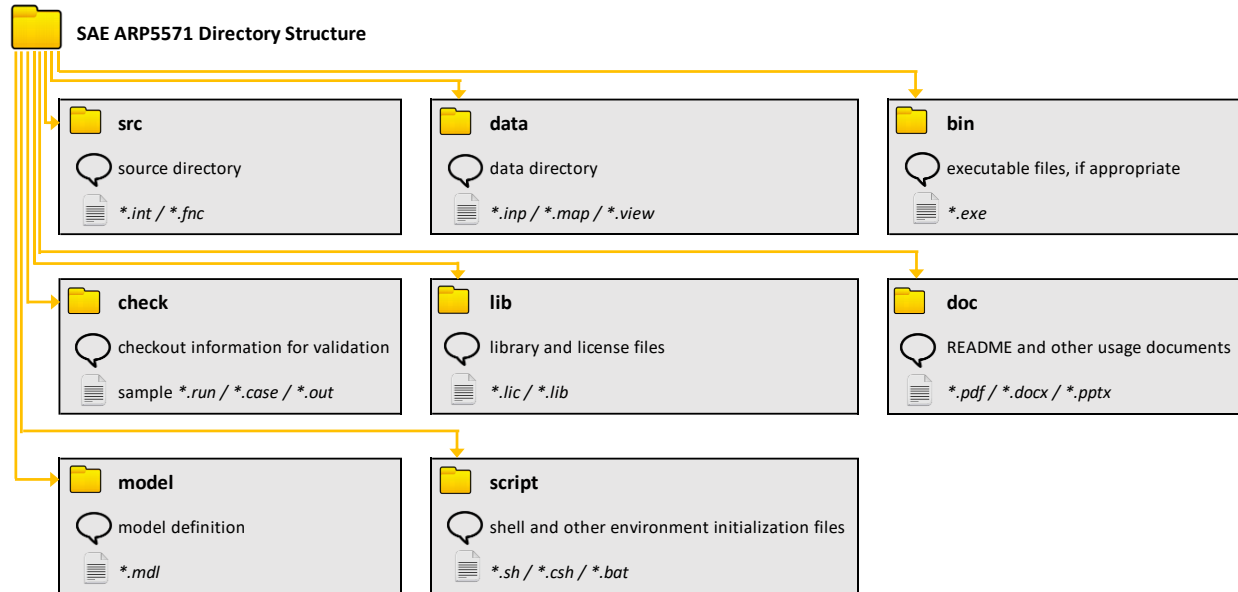


Figure 5. Recommended File Directory Structure According to SAE ARP5571

Figure 6 shows a simple example of a *.run file. In this case, the file is used to specify the thermodynamic package (allFuel), include the model with all of the elements and linkPorts() functions via the “burner.mdl” file, include some user-defined functions in the *.fnc file, identify a data viewer file for output data via the “printBurner.view” file, and include a user-defined file with solver parameters via the “burnerSolverSetup.inc” file. The image on the right-hand side of Figure 6 shows a section of the *.run file that defines a nested loop for evaluating the model over a range of lower heating values (LHV) and total temperatures at station number four (Tt4_in).

```

84 // -----
85 // Burner model from introductory training class
86 // -----
87
88 // Define desired thermo using preprocessor variables
89 #define THERMO allFuel
90
91 // Include the model
92 #include <burner.mdl>
93
94 // Include a user-defined function
95 #include <showJacob.fnc>
96
97 // Local includes for functions and viewers
98 #include <printBurner.view>
99 #include <burnerSolverSetup.inc>
100
101
102
103
104
105
106
107
108
170 // Set desired burner temperature
171 real Tt4_in { value = 2500.0; units = "R"; };
172
173 // Make array of burner temperatures
174 real Tt4Arr[] = {2200.0, 2500.0, 2600.0, 2700.0};
175 int iT4;
176
177 // Make array of LHV values
178 real LHVArr[] = {18400.0, 18000.0};
179 int iLHV;
180
181 // Loop through each LHV value
182 for ( iLHV=0; iLHV < LHVArr.entries(); iLHV++ ) {
183     Fus.LHV = LHVArr[iLHV];
184
185     // Loop through each burner temperature entry
186     for ( iT4=0; iT4 < Tt4Arr.entries(); iT4++ ) {
187         Tt4_in = Tt4Arr[iT4];
188         run();
189         printBurner.update();
190         CASE++;
191     } // end for loop through Tt4
192     printBurner.display();
193 } // end for loop through LHV

```

Figure 6. Typical Organization of a *.run File

One important aspect of the problem definition is the specification of the solver settings. NPSS is attractive because of the sophistication and configurability of the solver, which enables engineers to use the same model to solve a multitude of problems. For each problem that the engineer wishes to solve, the engineer has the ability to direct the solution by defining independent variables, dependent variables, and solver constraints.

An example of a solver independent/dependent pair definition is provided in Figure 7 on the left. In this case, the engineer is interested in knowing the fuel flow rate (Fus.Wfuel) required to achieve a specific burner temperature (F040.Tt). Therefore, the fuel flow rate is defined as an independent variable. Additional parameters are included in the independent object's definition to fine tune the solver. The burner temperature is identified as a dependent variable. Within the definition of the dependent object, NPSS is instructed to monitor the temperature at station 4 (F040.Tt) until it equals the user-input temperature specified by the "Tt4_in" variable.

In Figure 6, "Tt4_in" is evaluated over a range of user-input values. The NPSS solver iterates until a solution is determined. With each iteration, the solver updates the fuel flow rate and checks if the station 4 temperature equals the temperature demanded by the user. Executing the solution requires a simple command line entry shown in the right pane of Figure 7, which simply instructs NPSS to evaluate the "burner.run" file.

```
1 Independent ind_Wfuel {
2   varName = "Fus.Wfuel";
3   indepRef = "0.5";
4   dxLimit = 0.2;
5   dxLimitType = "FRACTIONAL";
6   perturbation = 0.01;
7   perturbationType = "FRACTIONAL";
8   description = "vary the fuel flow to achieve the desired burner temperature";
9 }
10
11 Dependent dep_F040Tt {
12   eq_rhs = "Tt4_in";
13   eq_lhs = "F040.Tt";
14   eq_Ref = "Tt4_in";
15   description = "desired burner temperature";
16 }
17
18 solver.addDependent("dep_F040Tt");
19 solver.addIndependent("ind_Wfuel");
20
```

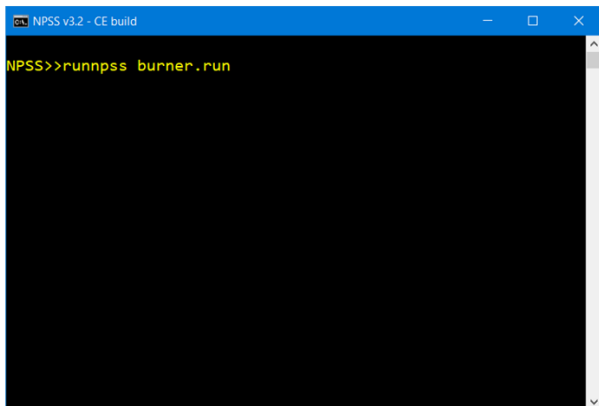


Figure 7. Example of a Solver Independent/Dependent Pair (Left) and Example Execution in the Command Window (Right)

This is a basic example of the overall problem setup and solution control. There are many more options available for controlling the simulation. The NPSS solver is robust and capable of handling many independent/dependent pairs in a single solution. Furthermore, multiple solvers can be used to improve the overall solution performance for models with sub-system assemblies. A few examples of solution modes include design, off-design, one-pass, steady-state, and transient. Design simulations are used to determine element performance characteristics required to meet the design objective; off-design simulations are executed to determine how the selected design will perform away from the design point; and transient simulations are used to study the system with time varying conditions such as power level, sudden load application, or even a pressure vessel blow down.

Viewing Output Data

Once the solution sequence is complete, there are a variety of ways to view the output data. The data can be sent directly to the screen or to a designated output file. Standard NPSS viewers enable the data to be presented in a column or row view, using the “CaseColumnViewer” and “CaseRowViewer” classes respectively. In Figure 8, the data from a sweep of burner temperatures is presented in a sequence of columns. Each column contains rows of data that were specifically identified for output in the problem setup. For these cases, it can be seen that increasing firing temperature leads to an increase in fuel flow rate. It is also possible to have the data sent directly to the command window for immediate feedback to the engineer.

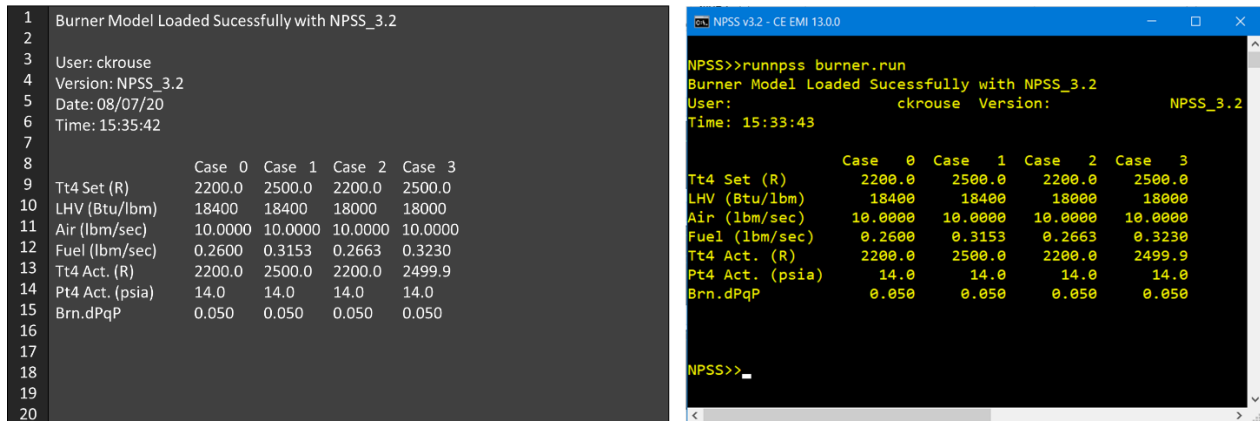


Figure 8. Output File from the “CaseColumnViewer” Class (Left) and Output Directed to the Command Window (Right)

The data in the column or row viewers can be imported into other plotting tools to generate graphs by the engineer. In addition to the column and row viewers, there is significant flexibility to define and format other data output summaries, such as an overall performance summaries, messages to the command line that can be used to monitor solution progress, csv files, etc.

Standard NPSS Thermodynamic Packages

The NPSS environment comes with a library of standard thermodynamic property packages that have been developed over many years by the Consortium Members. They are shown in Table 2. All of these packages are available to use in NPSS, and via the FPT property package, there is infinite ability for user-defined property definitions.

Thermo Package	Description
CEA	Implementation of the NASA chemical equilibrium code.
Janaf	Implementation of the National Institute of Standards and Technology gas properties prepared by Honeywell.
GasTbl	Package developed by Pratt & Whitney based on Therm, but adding humidity calculations and some chemical equilibrium capabilities.
AllFuel	Package developed by General Electric that contains gas properties and fuel properties.
FPT	Package used to define NPSS tables and/or functions that describe the thermodynamic properties of the fluid
REFPROP	REFPROP is not included in NPSS, but there is built in functionality to get properties from REFPROP if the user has the REFPROP code in the working directory (.dll and data files)

Table 2. Library of Standard NPSS Thermo Packages

Standard NPSS Elements

The NPSS environment comes with a library of standard elements that have been developed by the Consortium Members. These elements are continually updated and improved, and new ones are continually being developed by the Consortium. The standard NPSS elements are organized in the categories shown in Table 3, based on their intended uses.

Category	Description
AirBreathing	Typically used to model air-breathing engine cycles
Deprecated	FluidNetwork elements that have been deprecated, but are retained for backwards compatibility
FluidNetwork	Typically used to model generic fluid/thermal systems and liquid propulsion systems
General	Utility elements that have broad capabilities
Mission	Typically used to model and analyze aircraft missions

Table 3. Categories of the Standard NPSS Elements

AirBreathing Elements					
Ambient	Bleed	BleedOut	BleedOutInterstage	Burner	Compressor
ControlVolume	CrossOverValve	Cycle	Diffuser	Duct	Emissions
EngPerf	FlowDuplicator	FlowEnd	FlowStart	FuelSplitter	FuelStart
HeatExchanger	Inlet	InletStart	Instrument	InstrumentDuct	InverterValve
Load	Mixer	Nozzle	PortGroup	Propeller	Shaft
ShaftSpring	Slinger	Splitter	Turbine	Wall	

AirBreathing Subelements					
BurnEfficiency	CompressorRlineMap	dPdiffuser	dPqP	dPqPMach	FlightEnvelope
PropCT	RecoveryFactor	RecoveryRatio	TDay	ThermalMass	TurbinePRmap
Valve	WireCorrection				

Deprecated Elements				
FN_Duct	FN_DuctInertial	FN_FlowEnd	FN_FlowStart	FN_Leak
FN_Pipe	FN_PipeCf	FN_Pump	FN_Resistance	FN_ResistanceInert
FN_TMass	FN_Turbine	FN_Valve	FN_ValveHeadLoss	FN_Venturi
FN_Volume				

Deprecated Subelements		
FN_PumpMap	FN_TurbMap	FN_ValveHeadLoss

FluidNetwork Elements				
Leak	PlenumEnd	PlenumStart	PressureLoss	PressureLossInertia
Pump	ThermalMassVolume	TurbinePR	ValveHeadLoss	Venturi
Volume				

FluidNetwork Subelements	
PumpMap	TurbinePRdirectMap

General Elements
SocketGroup

Mission Elements				
AccelSegment	Airframe	AirframeAero	AirframeComponent	AirframeStack
AirframeState	AirframeWeight	AmbientState	ClimbSegment	CruiseSegment
DropSegment	ElementStack	EngineState	LoiterSegment	Mission
MissionSegment	MissionStack	PerformanceState	StartSegment	SubState
TurnSegment	VehicleState			

NPSS Example Models

NPSS is distributed with a number of example models. These models demonstrate how to do a variety of standard activities, such as instantiate and link elements, change thermodynamic packages, run transient simulations, switch between design and off-design, and create customer decks.

Model Name	Description
CDM00_Burner	Simplest model that contains only 4 elements
CDM01_Turbofans	A set of three unique turbofan models: fanjet, high-bypass turbofan, and low-bypass turbofan
CDM02_Thermal_Management	Generic aircraft thermal management system
CDM05_AC_Mission_Analysis	An example of how to couple the airframe and engine together in order to perform a mission analysis
CDM06_Turboprop	Generic turboprop model
CDM07_Model_Delivery	Two example approaches of how to secure NPSS models and create customer decks
CDM08_NPSS_Wrapper	Example approach of how to access external software by wrapping it and compiling it into a DLM

Table 4. Summary of Example Models Included with NPSS EMI

NPSS IDE

The NPSS IDE is a new NPSS tool that helps users visualize and manage file structures, modify NPSS files, and view model schematics. Currently, a beta release version is available to commercial customers upon special request. As with all the NPSS project areas, Consortium Members have access to the most recent bug fixes and the ability to request new features.

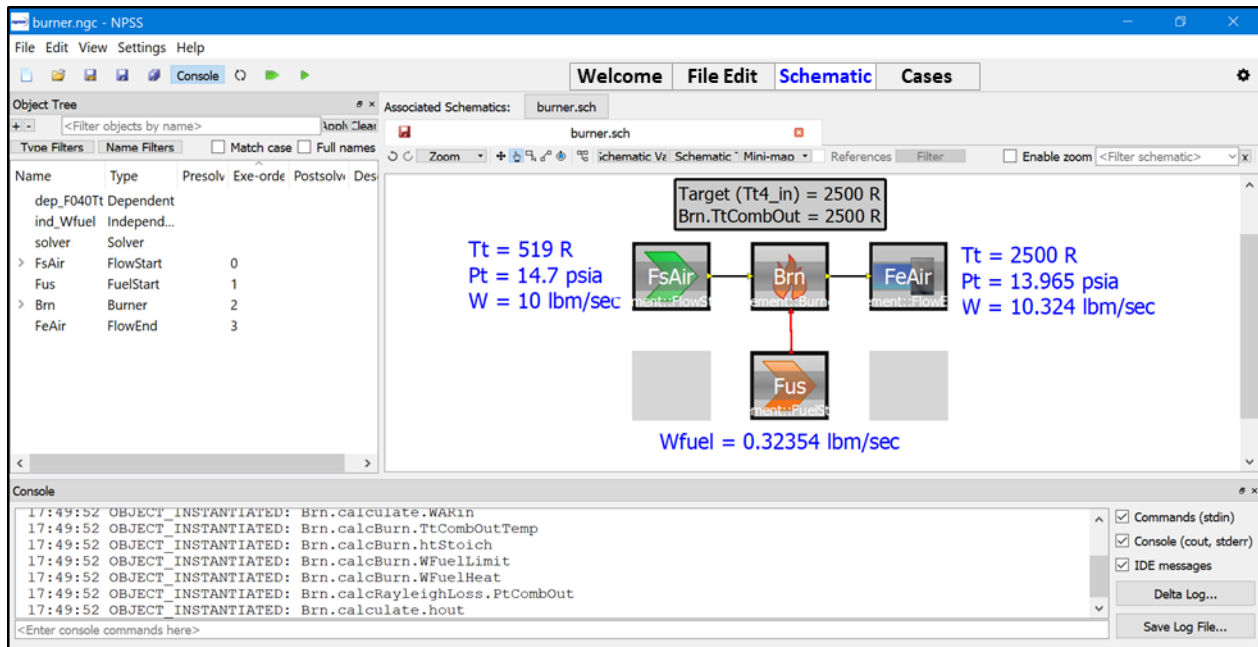


Figure 9. Example of a Screenshot from the NPSS IDE

How to Get NPSS

SwRI manages the NPSS Consortium on behalf of the Consortium members. Thus, you can purchase a NPSS license online via SwRI's [NPSS License page](#). Since NPSS is export controlled, the sale of NPSS licenses is restricted to countries not currently listed on the U.S. Department of Commerce Anti-Terrorism watch list.

For users or companies with significant experience using NPSS, or for companies with a large user base, Consortium membership may be beneficial. Members enjoy all the features of the commercial software, plus access to the full source code (a significant value for building the source to fit your own computing resources), unlimited NPSS licenses, opportunity to contribute towards defining future NPSS development efforts, and the legal right to sub-license NPSS to sub-contractors and vendors for the purpose of model sharing.

To contract SwRI for any NPSS modeling or development needs, for questions about the software, or for additional information about consortium membership, send an email to the NPSS Consortium Manager, [Charles Krouse](#), or call +1 210 522 5001.