



AEROSPACE STANDARD	AS5571™	
	Issued	2024-07
	Superseding ARP5571	
Gas Turbine Engine Performance Presentation and Nomenclature For Object-Oriented Computer Programs		

RATIONALE

Gas turbine engine manufacturers (suppliers) have long provided their customers with computer programs that simulate engine performance. Application manufacturers and others (customers) use these programs, often called models or simulations, in design studies, mission analysis, life cycle analysis, and performance prediction of their products. These models are used throughout the life of a product, from conceptual design through production, deployment, field use, maintenance, and overhaul. Communication between suppliers and customers is more productive and less error-prone if all engine models adhere to common guidelines with respect to presentation of data and interface with other computer programs. No guidelines or standards previously existed for object-oriented models.

This initial version as an aerospace standard supersedes ARP5571, which was a recommended practice for over 15 years. This is sufficient time for most suppliers to have become compliant with this document as a standard. This document is also more applicable to current technology than its Fortran predecessors and so requires higher recognition as a standard. Finally, due to the widespread use of object-oriented modeling, making this document a standard will help it to be more easily enforceable, both for contract purposes and business internal standards.

FOREWORD

Object-oriented programming, using languages such as C++, has introduced new features such as hierarchical object naming and object reuse. This provides improved capabilities not addressed in the existing standards. Sharing an entire modeling system, rather than individual programs, makes it desirable to standardize features of model architecture.

TABLE OF CONTENTS

1.	SCOPE.....	4
1.1	Purpose.....	4
1.2	Field of Application.....	4
2.	REFERENCES.....	4
2.1	Applicable Documents	4
2.1.1	SAE Publications.....	4
2.2	Related Publications	4
2.2.1	SAE Publications.....	5
2.2.2	Other Publications.....	5
2.3	Definitions	5
3.	MODEL INTERFACE.....	7
3.1	Model Usage Modes	7
3.1.1	Stand-Alone Mode	7
3.1.2	Application Program Interface (API) Mode	7
3.2	Model Types.....	7

SAE Executive Standards Committee Rules provide that: "This report is published by SAE to advance the state of technical and engineering sciences. The use of this report is entirely voluntary, and its applicability and suitability for any particular use, including any patent infringement arising therefrom, is the sole responsibility of the user."

SAE reviews each technical report at least every five years at which time it may be revised, reaffirmed, stabilized, or cancelled. SAE invites your written comments and suggestions.

Copyright © 2024 SAE International

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, or used for text and data mining, AI training, or similar technologies, without the prior written permission of SAE.

TO PLACE A DOCUMENT ORDER: Tel: 877-606-7323 (inside USA and Canada)
Tel: +1 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
http://www.sae.org

SAE WEB ADDRESS:
Full document details: SAE AS5571

Documents

For more information on this standard, visit <https://www.sae.org/standards/content/AS5571>

3.2.1	Engine-Specific Model	7
3.2.2	Shared-Environment Model	7
3.2.3	Restricted-Environment Model.....	8
3.2.4	Shared-Environment and Restricted-Environment Model Architecture	8
3.3	Relationship Between Model Usage Mode and Model Type.....	10
3.4	Application Program Interface (API)	10
3.5	User Interface.....	10
3.6	Model Execution Control	10
4.	NOMENCLATURE	10
4.1	General	10
4.1.1	Hierarchical Structure.....	10
4.1.2	Object Naming	11
4.1.3	Case Sensitivity.....	12
4.2	Multiple System Simulations	12
4.3	Component Instance Naming	12
4.4	Node Naming	14
4.4.1	Fluid Flow Nodes	14
4.4.2	Fuel Nodes.....	15
4.5	Parameter Naming	16
4.6	Units	16
4.7	Parameter Naming Guidelines.....	24
4.8	Option Switches	25
5.	PROGRAM CAPABILITIES	31
5.1	Supplier Provision for Custom Hook Socket Installation Interface	31
5.2	Customer Implementation of Custom Hook Socket Installation Interface	32
5.3	Alternate Approach in Evaluating Customer-Owned Logic.....	34
5.3.1	Customer Installation Objects Naming.....	34
5.3.2	Customer-Supplier Connectivity	34
5.3.3	Revisiting Example from 5.1 and 5.2 with the New “Cust” Object Philosophy	34
6.	PROGRAM STATUS INDICATION	35
6.1	Severity	36
6.2	Purpose.....	36
6.3	Category.....	36
6.4	Owner.....	37
7.	PROGRAM DELIVERY AND DOCUMENTATION	38
7.1	Model Identification	38
7.2	Model Information Feedback	38
7.3	Structure of Delivered Model	39
7.3.1	File Extensions.....	39
7.3.2	Directory Structure	40
8.	NOTES.....	40
8.1	Revision Indicator.....	40
APPENDIX A	Alternative Integer Switch Value Option Settings	41
Figure 1	Shared/restricted-environment architecture.....	9
Figure 2	Variable path to directory path analogy	11
Figure 3	Example of supplier inlet class code with hook socket provision	32
Figure 4	Example of customer inlet recovery class code used with hook socket	33
Figure 5	Example of customer inlet recovery class code used with hook socket	33
Figure 6	Example for customer-defined pressure recovery using Cust object	35
Table 1	Standard component instance names	13
Table 2	Fluid node naming examples	16
Table 3	Fuel node naming examples.....	16



Table 4	Standard attribute (parameter) names.....	17
Table 5	Qualifying prefixes and suffixes	25
Table 6	Standard switch variable option settings.....	26
Table 7	Severity code examples	36
Table 8	Error category indicators	37
Table 9	Error ownership codes	37
Table 10	Example file extensions	39

1. SCOPE

This document provides standards for several aspects of aero-thermal system performance modeling using object-oriented programming systems. Nomenclature, application program interface, and user interface are addressed with the emphasis on nomenclature. The Numerical Propulsion System Simulation (NPSS) modeling environment is frequently used in this document as an archetype. Many of the standards are derived from NPSS common practice. NPSS was chosen because it is an available product. The standards described herein may be applied to other object-oriented systems.

While this document applies broadly to any gas turbine engine, the great majority of engine performance computer programs have historically been written for aircraft propulsion systems. Aircraft and propulsion terminology and examples appear throughout.

1.1 Purpose

The purpose of this document is to define an object-oriented paradigm for the simulation of aero-thermal system performance, recommend nomenclature for objects and variables within such a simulation, and recommend practices for user interaction with such a simulation.

1.2 Field of Application

This document applies to aero-thermal system performance simulations using an object-oriented programming approach.

2. REFERENCES

2.1 Applicable Documents

The following publications form a part of this document to the extent specified herein. The latest issue of SAE publications shall apply. The applicable issue of other publications shall be the issue in effect on the date of the purchase order. In the event of conflict between the text of this document and references cited herein, the text of this document takes precedence. Nothing in this document, however, supersedes applicable laws and regulations unless a specific exemption has been obtained.

2.1.1 SAE Publications

Available from SAE International, 400 Commonwealth Drive, Warrendale, PA 15096-0001, Tel: 877-606-7323 (inside USA and Canada) or +1 724-776-4970 (outside USA), www.sae.org.

AS681 Gas Turbine Engine Performance Presentation for Computer Programs

AS755 Aircraft Propulsion System Performance Station Designation

AS5571 incorporates the recommendations of AS755 with respect to station designations. The nomenclature of AS6502 applies broadly to all forms of communication regarding propulsion system performance. AS681 allows “use of fully descriptive labels for variable identification” and provides that “In cases where symbols are to be substituted for fully descriptive labels, the list of symbols contained in AS6502 will be used ...”. The nomenclature in this document is more narrowly focused on the specific area of “fully descriptive symbols” to be used in the interface between users and computer performance simulations.

2.2 Related Publications

The following publications are provided for information purposes only and are not a required part of this SAE Technical Report.

2.2.1 SAE Publications

Available from SAE International, 400 Commonwealth Drive, Warrendale, PA 15096-0001, Tel: 877-606-7323 (inside USA and Canada) or +1 724-776-4970 (outside USA), www.sae.org.

AIR5509	Using Engine Test Data to Model Engine Performance
ARP4868	Function-Based API for Gas Turbine Engine Performance Programs
AS210	Definition of Commonly Used Day Types (Atmospheric Ambient Temperature Characteristics Versus Pressure Altitude)
AS4191	Gas Turbine Engine Performance Presentation for Computer Programs Using Fortran

2.2.2 Other Publications

NPSS Consortium, www.npssconsortium.org

NPSS User Guide

NPSS ARP4868 API User Guide

2.3 Definitions

API: Application program interface. A protocol agreed to between the supplier and the user for interaction between the customer's computer program and the supplier's engine performance simulation, when the supplier's program is operating as a subprogram.

ATTRIBUTE: Characteristic of an object. Similar in concept to a variable in Fortran - efficiency is an attribute of a compressor, for example. But variables themselves have attributes in object-oriented programming; value, description, and units are all typical attributes of a variable.

CLASS: The code that forms the pattern or blueprint for an object type.

COMPONENT: See ELEMENT.

CUSTOMER: The person or organization that receives and executes the engine simulation program. Same as USER.

ELEMENT: The primary building block of a model. Somewhat analogous to a Fortran subroutine. An element may represent either a physical engine component or a process. Same as COMPONENT.

ESI: Engineering Status Indicator. An 8- or 9-digit integer conveying error, warning, and message information. This indicator is an extension of the 4-digit NSI (Numerical Status Indicator) Concept defined in AS4191.

GUI: Graphical User Interface.

HOOK SOCKET INTERFACE: A specific object type, provided by the supplier, which allows the user to plug into (or hook up to) certain sockets in the model. Hook sockets allow the user to provide their own custom code for installation effects.

INSTANCE: A particular occurrence of an object type. For example, CmpH might be one instance of an object type called Compressor. There can be multiple instances of the same object type.

INSTANTIATE: To create an instance of an object type.

INTERPRETED CODE: Uncompiled code that is interpreted and executed at run time.

MEMBER FUNCTION: A function (an action) that belongs to an object.

MODE: Execution method for a model that controls a set of features and setup required for a specific type of analysis, such as Steady State, Transient, or Data Reduction modes.

DATA REDUCTION MODE (DR): An execution mode that represents a process used to match a model to measured data. Refer to AIR5509.

STEADY STATE MODE (SS): An execution mode where all model parameters are thermodynamically stable, and there is no time-dependency.

TRANSIENT MODE (TR): An execution mode where thermodynamic parameters and/or variable geometries change as a function of time and whose current values are a function of the history of previous answers.

MODEL: (1) A computer program together with the appropriate input that simulates the aero-thermal performance (and possibly other characteristics) of a particular system design. (2) A specific member of a gas turbine product line, e.g., PW120 or CFM56-7.

NODE: A connection between components or elements that passes fluid, fuel, or other properties. Same as PORT.

NPSS: Numerical Propulsion System Simulation. An object-orientated simulation development environment developed jointly by NASA and industry.

OBJECT-ORIENTED: "Object orientation means that the program's structure revolves around objects of various types, each of which has a set of attributes accessible by other objects, and certain functions - actions particular to the object - that can be performed at the request of other objects. An object may be a simple entity such as a variable, or a complex entity such as a solver (an object which performs the necessary numerical iteration)." (Refer to NPSS User Guide.) Java, Ada, and C++ are object-oriented languages.

OBJECT: An instance of a class. Objects may be instantiated within other objects (subobjects). Subobjects (child objects) are said to belong to the object containing them (parent object).

PARAMETER: An entity that is used to describe the condition or state of a particular object in the engine or engine program. Example: Amb.alt_in, CbldECS.W.

CALCULATED PARAMETER: Any engine program parameter with a value calculated by the engine program. Often, these are referred to as output parameters. The term calculated is used to avoid ambiguity with input parameters that may also be available for and displayed as output.

INPUT PARAMETER: A parameter that can be set by the customer and which the engine program should not overwrite.

INTERMEDIATE PARAMETER: A parameter that provides the target or demand value for the model solver. Its value may come from the corresponding input parameter, from a user's hook socket instantiation, or from a calculation in the supplier's code.

PORT: A connection between components or elements that passes fluid, fuel, or other properties. Same as NODE.

SCOPE: Scope is the Object/subobject/... chain of ownership of a variable, function, or other object. Analogous to directory path in a file system.

SOCKET: A particular type of object that acts as a stub within the code. Each socket can accept only one object of a specific, predetermined type.

STATION: A specific location of thermodynamic interest to the model.

SUBELEMENT: An object that is the child of an element or of another subelement.

SUBOBJECT: A more generic term than subelement to describe an object that is the child of another object.

SUPPLIER: The organization that creates and delivers the engine simulation program.

SWITCH: An input parameter with a predefined list of allowable options that selects among alternate paths through the program logic.

USER: See CUSTOMER.

3. MODEL INTERFACE

This section addresses user interface and Application Program Interface (API) issues, which do not concern the internal architecture of the model itself. These issues need to be understood by both the supplier and the customer so that an interface protocol beneficial to both may be agreed upon. This document provides a basis for identifying the issues to be discussed. Elements of this standard shall only be waived by mutual consent of the program supplier and customer.

3.1 Model Usage Modes

Models are used in two fundamentally different ways or modes. Which mode is used is determined by whether the model supplier or the model user is responsible for (1) executive control of the simulation and (2) the formatting of input and output.

3.1.1 Stand-Alone Mode

For this mode, the program provided by the supplier includes a main program that is used as delivered. There is no API, and the supplier is responsible for executive control of the simulation and for its input and output. The user controls execution through a user interface.

3.1.2 Application Program Interface (API) Mode

For this mode, the program provided by the supplier is used as a subprogram. The model user provides the main program that governs executive control of the simulation and provides for input and final output. The engine program is thus an application that interacts with the user's software through a mutually agreed API. The supplier generally provides a sample main program (in source code form) for checkout purposes, but the user is free to modify or replace this program.

3.2 Model Types

Model suppliers can provide their models in several different forms. Three common types of models are described in this section.

3.2.1 Engine-Specific Model

The supplier provides a complete program to the user, together with any input files required by the program. The program cannot be modified for other uses; it can only produce results for the specific engine configuration for which it was written. To provide an updated model, the supplier must generally provide a new program as well as new input files. In principle, the supplier could provide program source to the user. Historically, however, it has been more common for the supplier to compile and link an executable for the user's platform.

3.2.2 Shared-Environment Model

The supplier and customer use the same environment kernel to develop and run models. There will likely be differences in the full computing environment of the two companies due to, for example, different configuration management systems. However, the supplier and customer share the essential software (the simulation program itself and any supporting software required by that program) used to produce simulation results. The essential characteristic of shared-environment models is that the supplier need only provide a model definition to the customer, not the complete simulation software. The model definition quite often will consist of one or more files read as input by the simulation program and may consist of special software modules (either in source or compiled form) called by the simulation program. Ideally, the supplier sends the very same model definition files as used by the supplier internally. This situation is analogous to one user providing a Microsoft Excel workbook model to another Excel user. The supplier needs only to send an Excel input file and any macros used by that file. Both supplier and customer share the computing environment kernel (Excel) required to use those files.

3.2.3 Restricted-Environment Model

Programs like NPSS make a third model type possible that stands between the two previously described. As stated above, with shared-environment models, the supplier and customer need only share the environment kernel required to run models.

However, another feature of a restricted-environment model is the ability of the supplier to selectively restrict the user's access to certain aspects of the model. This is done to allow the supplier to protect information in order to stay in accordance with various rules, such as government security regulations, export restrictions, proprietary agreements (or lack thereof), or any other reasons mutually agreed to by the user and supplier.

A restricted-environment model can be created to model many different engine configurations by loading the appropriate input files and/or files containing supplementary code. Some such files may be encrypted for privacy, but others may be open for user modification or replacement. In this way, the user may be able to override agreed-to parts of the simulation as provided by the supplier or add his own calculations.

For example, a vehicle manufacturer could receive a restricted-environment program from an engine supplier together with the files defining a specific engine and use the program input capabilities to add installation effects and even construct a complete vehicle simulation. When the engine model needs to be updated, the engine supplier provides a new set of input files tailored to the program sent earlier. The engine supplier could also provide input files defining a completely different engine as long as they are designed to work with the original program. From the user's point of view, this arrangement is much like a shared-environment model.

3.2.4 Shared-Environment and Restricted-Environment Model Architecture

Shared-and restricted-environment models have a great deal in common. Both types of model shall be delivered to support the structure shown in Figure 1.

This architecture provides great flexibility in integrating the engine model with the application if desired. The intent is to modularize the different aspects of an integrated engine/vehicle installed performance analysis such that each aspect can be coded, maintained, and even operated independently of all the others. Models need not be hard-coded specifically for one installation.

The "Program Kernel" is defined in this context as the shared environment (e.g., the NPSS binary) that interprets all of the objects and solves the system at the desired conditions.

The "Supplier Engine Model Definition Files" are the files delivered by the supplier. Collectively, these files will include any source code that is not part of the kernel and any data or input files required to define the engine configuration. The code will generally include provisions for hook sockets, which allow users to provide their own code for installation effects such as inlet loss.

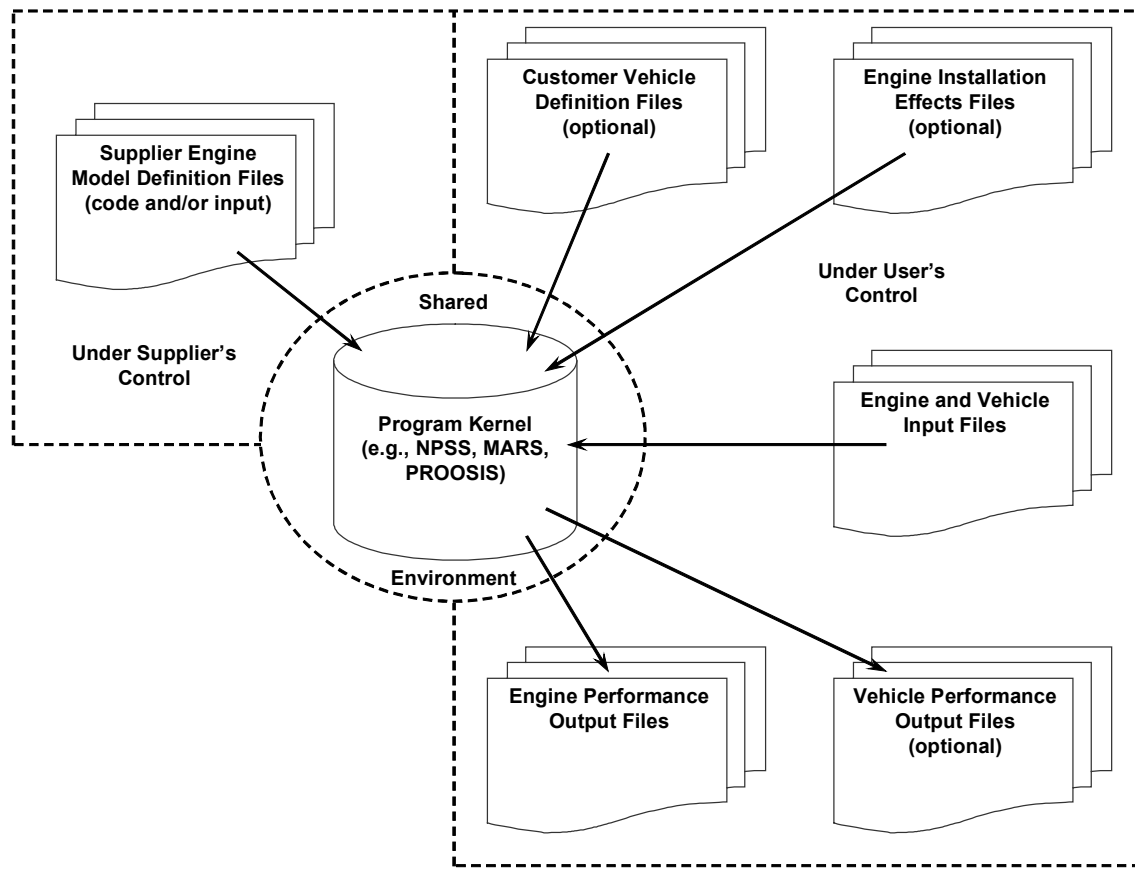


Figure 1 - Shared/restricted-environment architecture

“Customer Vehicle Definition Files” are optional files, compatible with the program kernel, which can be used to model the customer’s application or, more likely, subsystems of the application. These files could contain transmission or propeller models, for example.

“Engine Installation Effects Files” provide the user owned hook socket instantiations, which link up to predetermined sockets in the engine model.

“Engine and Vehicle Input Files” contain the typical inputs to control vehicle and engine operating condition: altitude, ambient conditions, bleed demand settings, etc. This category may also include files to control model output content and format.

Although a model may be delivered with a default output format, output content and format should be under the user’s control.

With this architecture, it is possible to replace any of the various components of a configuration without modification to any other. This is demonstrated by the following use case:

An engine manufacturer has provided a simulation for a family of engine models. The simulation is in regular use for prediction of engine performance on a specific vehicle configuration.

A preliminary design task has begun, using the existing engine simulation to evaluate changes in the aircraft inlet, bleed, horsepower extraction, exhaust nozzle, and force accounting systems.

All of the aircraft systems being modified interact with the engine model through predetermined hook sockets in the simulation code.

The customer can quickly evaluate the impact of the proposed system changes on engine and vehicle performance without any changes to the supplier's engine simulation code. He can also evaluate the performance of other members of the same engine family in the modified installation.

3.3 Relationship Between Model Usage Mode and Model Type

Engine-specific programs can be delivered either in stand-alone or API mode. With shared-environment models, supplier and customer are generally using the same environment kernel in stand-alone mode. If the customer wants to call the simulation from another main program, it is the customer's responsibility to create an application version of the simulation program. Restricted-environment models may be supplied in either stand-alone or API mode, as agreed between the supplier and customer.

3.4 Application Program Interface (API)

The purpose of the API is to provide communication between the customer's main program and supplier's application program. This communication is used to control the execution of the application program and to exchange data with the main program. It is important for the supplier and customer to agree on the API, such as the functions described by ARP4868.

3.5 User Interface

A user interface is an interface by which a user communicates with a program, usually being run in stand-alone mode. Command-line interfaces and Graphical User Interfaces (GUIs) are two broad classes of user interface types.

Related to the concept of user interface is program input format. Input files prepared by a user are also a means by which a user controls the execution of a simulation program. Input format is applicable to stand-alone programs and programs run through an API. Output format is also important to all model types and modes. This document will address some issues of user interface, input format, and output format, especially since they take on increased importance in shared-environment models and restricted-environment models.

3.6 Model Execution Control

A model may have multiple modes of execution to account for various types of modeling and analysis behaviors. Two common execution modes are Steady-State (SS) and Transient (TR). A model shall contain appropriate execution control mechanisms for any required modes of operation, and the current execution mode shall be clearly represented by a variable or switch that is visible in the model.

4. NOMENCLATURE

This section makes detailed recommendations for descriptive names to be used in object-oriented engine models. The lists of parameter and object names in this section are not to be construed as requirements to provide customer access to their values. As has always been the case, suppliers and customers should agree in advance to a mutually acceptable degree of openness in their information exchange, subject to the strictures of government classification, export control, proprietary agreement, and need to know.

4.1 General

Object-oriented nomenclature is quite different from Fortran-based nomenclature. Name length is not restricted; fully descriptive labels are encouraged, as are mnemonic abbreviations for names that are frequently typed. Mixed case names are common because they improve readability of code and input.

4.1.1 Hierarchical Structure

The structure of object-oriented code is hierarchical; objects are the parent or child of other objects. Specifying the full name of an object or variable requires specifying the chain of parent-child relationships leading to the object. This chain of ownership is known as the path of an object. This path is very much analogous to the directory path required to fully describe a file in a directory structure. See Figure 2. Whereas directory names are delineated by a slash (/), object names are typically delineated by a period, or "dot" (.).

Object-oriented Variable Naming is Analogous to Computer File Naming

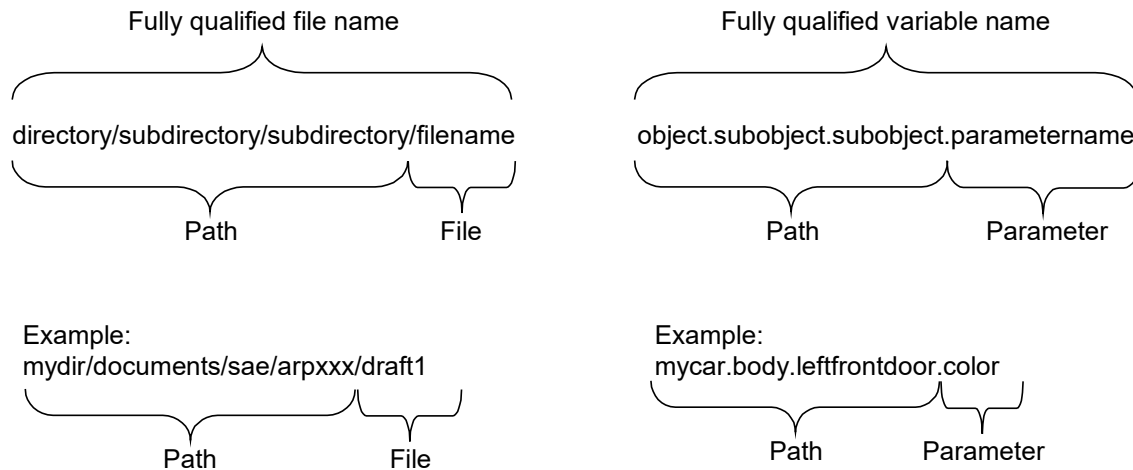


Figure 2 - Variable path to directory path analogy

There are several important details about path that are important to understand.

- Path is relative, just as it is for directories. Any path information that appears in the code for an object is relative to that object.
- A name may be used without any path information in the object where it is created and in all children of that object. The part of the model where an object name can be used without specifying a path is called the object scope. Object names must be unique within their scope.
- Some models may use '.' for other purposes. In that situation, use of another character is permitted to delineate scope.
- Some objects belong to the top level of the model and require no path in their names. The leading dot is inferred.
- A common name such as T3 might become something like F030.Tt, which is short for Flow Station 030 Total Temperature.

Parameter is not necessarily the lowest level of the hierarchy; parameters, in turn, own attributes such as their value, description, units, and input/output status.

4.1.2 Object Naming

It is desirable to minimize the amount of path information needed when dealing with common interface variables between the engine and the application. It is also extremely desirable for both the path and parameter names to be common across a wide range of models from different suppliers. To that end, the nomenclature that follows establishes naming conventions for several categories of objects:

- Top level objects representing components or processes that customers frequently interact with, such as inlets, compressors, or bleeds.
- Attributes of these objects, such as input, output, and switch parameter names and allowable options. (Switches are akin to SIM and SERAM in Fortran models.)
- Nodes, which are the connection points between components. Node types are fluid (the equivalent of flow stations), fuel, and mechanical.

Suggestions for naming new objects not anticipated in this document are offered in 4.2 for components and in 4.6 for parameters.

4.1.3 Case Sensitivity

Fully descriptive parameter names tend to be quite long. Readability is enhanced if uppercase and lowercase letters are used in predictable ways. To take advantage of mixed-case parameter names, the program kernel must obviously be case-sensitive. The case conventions of the sections that follow shall be observed when writing code for any case-sensitive system.

4.2 Multiple System Simulations

Higher fidelity models may make use of multiple simulations operating in concert in a single execution. Such examples may include:

- Multiple Engine Models
- Engine + Thermal Management System
- Aircraft Performance + Engine Model

In order to facilitate integration of these various system models in a single simulation, the following object names are recommended for use. In instances where multiple versions of a single type of model are used, adding a numerical identifier is recommended.

Simulation Type	Name Base	Recommended Instances	Instance Description
Engine Performance	Eng	Eng	Engine Simulation
		Eng01, Eng02	Engine Simulation 1 & 2
Aircraft	AC	AC	Aircraft Performance, Installation Effects
Thermal Management	TMS	TMS	Thermal Management System

4.3 Component Instance Naming

Using standard instance names improves communication about models between suppliers and customers. Standard naming allows a single application program to be used with different models. Table 1 provides examples of instance names for common components and processes. Each instance name is constructed from a name base and an appended identifier. The name base indicates generically what type the component is, e.g., Cmp for Compressor. The identifier makes the name specific and unique within a given engine model. For example, a model might have CmpFan and CmpH modules. Both the name base and the identifier shall have only their initial letters in upper case. Names of components not anticipated in this table shall be constructed using these guidelines.

It should be noted that, in models conforming to this document that are of different engine configurations or come from different suppliers, identically named components may not have identical roles in the cycle. As an example, in one model, CmpH might represent the entire gas generator compressor, while in another CmpH might represent only the centrifugal stage of a gas generator compressor comprising several axial stages and a centrifugal stage on the same shaft. Also, a component in the context of this table may be a single object or a collection of objects. CmpH might represent solely the compression process between two fluid nodes, or it might also include additional processes such as bleed and exit diffusion. All these examples conform to this standard. Communication and agreement between supplier and customer are critical.

Table 1 - Standard component instance names

Component or Process	Name Base	Recommended Instances ⁽¹⁾	Instance Description
Ambient	Amb	Amb	Freestream conditions
Bleed	B	----(2,3)	
Burner	Brn	BrnPri	Primary burner
		BrnAug	Augmentor (afterburner)
Compressor	Cmp	CmpFSec	Fan OD or secondary (split fan)
		CmpFPri	Fan ID or primary (split fan)
		CmpFan	Fan (not split)
		CmpL	Low pressure compressor
		Cmpl	Intermediate pressure compressor
		CmpH	High pressure compressor
		CmpLd	Load compressor
Control	Ctrl	Ctrl	Engine control
Customer Bleed	Cbld	CbldECS	Customer bleed - environmental control syst.
		CbldAI	Anti-ice customer bleed
		CbldWAI	Wing anti-ice customer bleed
		CbldCAI	Cowl anti-ice customer bleed
		CbldPreC	Precooler bleed
Customer Drag	Cdrag	CdragSt	Stream Tube Drag
		CdragIn	Inlet Drag
		CdragNoz	Nozzle boat tail Drag
		CdragPod	Engine Pod Drag
Customer Power Extraction	Cpwr	CpwrL	Power extraction from low spool
		CpwrI	Power extraction from intermediate spool
		CpwrH	Power extraction from high spool
Cycle Summary	Cycle	Cycle	Overall cycle parameters, e.g., bypass ratio
Duct	D	----(2)	
Duct Instrumented	Di	----(2)	
Engine Interface	Eint	Eint	Engine Interface
Flow-duplicator	Fdup	FdupPri	Primary stream flow duplicator
		FdupSec	Secondary stream flow duplicator
Flow-end	Fe	FePri	Primary stream flow end
		FeMix	Mixed stream flow end
		FeSec	Secondary stream flow end
Flow Restriction	Fr	FrValve	Minimum restriction due to a valve
		Fr228	Minimum restriction at plane 228
		FrSeal	Minimum restriction due to seal leakage
Flow-start	Fs	FsEng	Total engine stream flow start
Fuel-splitter	Fusplt	FuspltEng	Splits fuel output of FusEng
Fuel-start	Fus	FusEng	Single fuel start for engine ⁽⁴⁾
		FusPri	Multiple fuel starts - primary ⁽⁴⁾
		FusAug	Multiple fuel starts - augmentor ⁽⁴⁾
Gear	Gear	GearFan	Gearbox driving fan
		GearProp	Gearbox driving Propeller
Heat Flux	Hf	Hf012	Inlet anti-ice recirculation heat
		HfVent	Heat lost to a vent
Heat exchanger	Hx	----(2)	
HeatSoak	Hs	Hs040	Heat soak through the combustor liner
		HsCmpH	Heat soak through a compressor
Inlet	In	InEng	Engine inlet (non-split flow)
		InPri	Primary inlet (split flow)
		InSec	Secondary inlet (split flow)
Mixer	Mix	MixExh	Engine Exhaust stream mixer
Nozzle	Noz	NozPri	Primary stream nozzle
		NozSec	Secondary stream nozzle
		NozMix	Mixed stream nozzle

Table 1 - Standard component instance names (continued)

Component or Process	Name Base	Recommended Instances ⁽¹⁾	Instance Description
Propeller	Prop	Prop	Propeller
Performance	Perf	Perf	Engine performance (base-Uninstalled)
		PerfSt	Stream-tube-thrust engine performance
		PerfPod	Podded-thrust engine performance
		PerfMin	Minimum-engine performance
Power Setting	Pset	Pset	Power Setting
Shaft	Sh	ShL	Low Pressure shaft
		ShLG	Low Pressure shaft - geared
		ShI	Intermediate Pressure shaft
		ShH	High Pressure shaft
		ShP	Power turbine shaft
Splitter	Splt	SpltFan	Fan Splitter (Pre or Post)
Turbine	Trb	TrbL	Low Pressure turbine
		TrbI	Intermediate Pressure turbine
		TrbH	High Pressure turbine
		TrbP	Free Power Turbine

- ⁽¹⁾ For components not described in the table, or for components that do not have recommended instances, the identifying suffix may be a string or a station label. If the identifier is a string, its first letter is capitalized with following letters in lower case unless they start a new word. If the instance identifier is a station number, it shall conform to the "Alternate System for Reduced Ambiguity" of AS755.
- ⁽²⁾ The identifying suffix for the indicated components may be either a station designation or a commonly used abbreviation. Examples: B030, D125, DEGV, and Di030.
- ⁽³⁾ The Bleed object does the bookkeeping of bleed flow with respect to the main flow. The bleed object may be identified by either its entrance or its exit station. In typical usage, bleed extraction processes shall be named after their entrance stations, and the bleed return processes are named after their exit stations. For example: B030 (entrance station 030, exit station 031) might model bleed extracted after the compressor, and B041 (entrance 040, exit station 041) bleed returned to the main flow ahead the turbine rotor.
- ⁽⁴⁾ Use FusEng when a single fuel source is used. This may provide fuel to a single burner or, when used with a fuel splitter, provide fuel for two burners. Use FusPri and FusAug if two separate fuel sources are modeled, one for the main burner and one for the augmentor (afterburner). If two or more fuel sources are used for other reasons, select meaningful names - FusJP4, FusJP5, FusTank1, FusTank2.

4.4 Node Naming

Nodes provide access to variables that link components. Like all variable names, the node variable names consist of two parts: the path and the parameter name. The path is simply the node name. The node name begins with a prefix that defines its type:

- a. Fluid flow node names begin with F
- b. Fuel node names begin with Fu

4.4.1 Fluid Flow Nodes

A fluid flow node provides the connection for flow passing from one component to another. A component may have multiple fluid input ports and fluid output ports. By definition, a fluid node connects one output port to one input port of another component. The fluid flow node name is the F prefix followed by a station number. The standard for selecting station numbers are as follows (refer to AS755):

- a. The only one-digit station allowed is 0; all other stations require three digits.